Aceleración de la fase de inferencia en Redes Neuronales Profundas con dispositivos de bajo coste y consumo

Inference acceleration in Deep Neural Networks on low-cost and low-consumption devices

Juan Mas Aguilar

MÁSTER EN INGERNIERÍA INFORMÁTICA. FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

Trabajo Fin de Máster en Ingeniería Informática
Convocatoria de febrero de 2020
Calificación Obtenida: 10 - Sobresaliente

07/02/2020

Directores:

Guillermo Botella Juan
Alberto Antonio del Barrio García

Autorización de difusión

Juan Mas Aguilar

10/01/2020

El abajo firmante, matriculado en el Máster Ingeniería Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "Aceleración de la fase de inferencia en Redes Neuronales Profundas con dispositivos de bajo coste", realizado durante el curso académico 2019-2020 bajo la dirección de Guillermo Botella Juan y Alberto Antonio del Barrio García en el Departamento de Arquitectura de Computadores y Automática, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Resumen en castellano

Este trabajo profundiza en el ámbito del *Deep Learning* o aprendizaje profundo. Analizaremos el estado del arte y nos centraremos en las tecnologías de aceleración de la fase de inferencia con dispositivos de bajo coste. Para ello, se ha realizado una aplicación de reconocimiento facial en vídeo sobre el que se han tomado medidas de consumo, rendimiento y escalabilidad. Para el despliegue de la aplicación, se han utilizado dos modelos distintos de red neuronal basados en Tensorflow, el Intel© OpenVINO™Toolkit y los dispositivos Intel© Neural Compute Stick 2 para la aceleración de la inferencia.

Con todo ello, se ha podido comprobar la utilidad de los dispositivos NCS2 para la aceleración de la fase de inferencia consiguiendo tiempos cercanos al *real-time* y mejorando el rendimiento de la red neuronal en escenarios de alta demanda.

Palabras clave

Tensorflow, Python, OpenVINO, Inferencia, Redes Neuronales, Intel, Neural Compute Stick, Reconocimiento Facial, FaceNet, Aprendizaje Profundo

Abstract

This paper deepens in the study of the acceleration of the inference phase using low-cost and low-powered devices. We we will analyze the state of the art and we will focus on the inference acceleration technologies used over embedded systems. For this purpose, a video face recognition application has been developed and measurements of consumption, performance and scalability have been taken. In order to deploy this application, two different Tensorflow-based CNNs have been used, optimized and deployed using the Intel© OpenVINO™Toolkit and various Intel© Neural Compute Stick 2.

With this, the usefulness of these kind of devices could be proven in order to accelerate the inferences of a general, not ad-hoc CNN, achieving promising frame rates in almost real time restrictions and increasing the performance in demanding situations.

Keywords

Tensorflow, Python, OpenVINO, Inference, Neural Networks, Intel, Neural Compute Stick, Face Recognition, FaceNet, Deep Learning

Índice general

Índice de figuras

Índice de tablas

Agradecimientos

A mis tutores, por todas las correcciones y el contiuo apoyo y confianza que han mostrado en mí. A los amigos, por compartir todas mis experiencias a lo largo del grado y del máster. Y por último, a mi familia, que siempre ha estado ahí para apoyarme.

Capítulo 1

Introducción

Históricamente, las aplicaciones de inteligencia artifical se veían límitadas por la capacidad de cómputo disponible en ese momento. Una prueba de ello es la existencia de las redes neuronales. La primera noción de perceptrón se definió en 1958 de la mano de F.Rosenblatt [1] como una unidad que recibía una serie de entradas y aplicaba una función sobre esos datos. Más tarde, en 1965, se definíria el concepto de perceptrón multicapa. No obstante, aunque se fuera consciente de la potencia de este modelo probabilístico, la comunidad científica y técnica descartó su uso debido a la enorme cantidad de datos que se debían proveer a la red neuronal para que esta alcanzara un cierto grado de precisión en sus predicciones.

Actualmente sí tenemos la capacidad de cómputo suficiente para manejar estas ingentes cantidades de datos. Las ramas de investigación en Big Data (conjunto de técnicas que permiten manejar de forma eficiente estos datos) y HPC (High Performance Computing) han permitido poner toda esa información a disposición de los algoritmos de inteligencia artificial.

Aquí es donde entran en juego las ramas del Machine Learning y, más concretamente, el Deep Learning. De forma genérica, se suele definir el Machine Learning cómo el uso de algoritmos para procesar datos, extrapolar conclusiones de ellos (proceso que llamamos aprendizaje) y, por último, realizar predicciones sobre nuevos datos. Por contra, el Deep Learning es un subconjunto de técnicas dentro del Machine Learning. En ellas, los mismos algoritmos utilizados en Machine Learning son combinados entre ellos en multitud de capas

consiguiendo así que cada una de ellas arroje una perspectiva distinta sobre los datos que se le han provisto. De entre estas técnicas destacan las ya mencionadas redes neuronales.

En este trabajo, profundizaremos en esta rama de la inteligencia artificial e intentaremos arrojar luz sobre nuevas perspectivas y soluciones que se encuentran actualmente en el mercado.

1.1. Motivación

Actualmente nos encontramos en plena fiebre de la Inteligencia Artificial (IA), todas las empresas y todos los productos aplican de una u otra forma IA. Adicionalmente, es un hecho que transpasa las fronteras de la comunidad científica y que, cada vez más, llega a los consumidores finales como un concepto novedoso y revolucionario.

Técnicamente hablando, hemos alcanzado un punto en el que somos capaces de entrenar redes neuronales con miles de millones de datos, consiguiendo porcentajes de predicción o identificación en ocasiones superiores a las de un ser humano. Es por eso que el foco ya no reside en el entrenamiento de las redes neuronales facilitado por el potencial de las GPU modernas, sino en la minimización de los parámetros de aprendizaje o tiempos de inferencia.

Muchos de las nuevas barreras de la investigación en este ámbito (coches autónomos, androides, sistemas de seguridad domóticos), basan su funcionamiento en tiempos de respuesta mínimos ante un estimulo. Hasta ahora, la forma más eficaz de conseguirlo era conectando con un servidor de procesamiento de datos alojado en la nube (lo que conseguía minimizar el hardware necesario así como su precio). No obstante, en ocasiones, los tiempos de latencia de comunicación entre el dispositivo que recibe los datos y el servidor en la nube son del todo inasumibles. Un caso claro de esta problemática sería el software que maneja un coche automático en caso de un accidente inesperado.

Por otra parte, introducir la inferencia en sistemas empotrados deriva en una clara limitación del rendimiento dado que este tipo de sistemas disponen de una potencia de cálculo muy limitada. Este hecho puede ser asumible en algoritmos sencillos, pero en casos

como el reconocimiento facial, donde el número de caras afecta directamente al rendimiento (siendo este un parámetro altamente escalable en escenarios no controlados), no resulta suficiente y se evidencia la necesidad de introducir algún agente o componente sobre el que derivar el cálculo.

Es por esta razón que se dedican muchos esfuerzos para desarrollar tecnologías que aceleren la fase de inferencia de una red neuronal, de forma que se puedan desplegar directamente sobre los dispositivos que recogen los datos o junto a ellos. Esto permite minimizar la latencia a tiempos despreciables. En este trabajo, se tomará una de esta tecnologías y se realizarán distintos tipos de mediciones para comprobar su funcionamiento y rendimiento en entornos reales.

1.2. Objetivos

El objetivo planteado consiste en desarrollar una aplicación de reconocimiento facial que permita medir las posibilidades de escalabilidad horizontal de dispositivos de bajo coste que puedan asumir tareas de procesamiento complejo. Para ello, se comprobará su rendimiento, el coste tanto a nivel de mercado como de consumo y sus posibilidades en escenarios situados en la frontera (Edge Computing).

A continuación se enumeran los objetivos concretos:

1. Desarrollar una aplicación de reconocimiento facial utilizando Python y el toolkit Intel© OpenVINO™acelerando la inferencia mediante los Intel© Neural Compute Stick 2.

2. Desplegar la aplicación tanto en un dispositivo de potencia media actualmente (Intel i7 de 4ª generación) como en un dispositivo de bajo coste y consumo (Raspberry Pi 3B+ y Raspberry Pi 4). Tomar mediciones de rendimiento, consumo y escalabilidad.

3. Analizar los datos obtenidos para extraer conclusiones sobre el tema planteado. Comprobar la eficacia de las tecnologías escogidas y sus posibilidades en casos de uso en la

frontera frente a tecnologías de mayor coste y potencia.

Para ello se desarrolló el siguiente plan expuesto en el mismo orden que los apartados de esta memoria.

1. Comprobar el estado del arte en reconocimiento facial, redes neuronales y Edge Computing con el objetivo de orientar la investigación y seleccionar la técnica que mejor se adecue a los objetivos planteados. Esta etapa se desarrolla en el capítulo 2.

2. Desarrollo de la arquitectura hardware y software de cara al despliegue e implementación de la aplicación. Esta estapa es desarrollada en el capítulo 3.

3. Una vez creada la arquitectura, desarrollar una aplicación que se despiegue sobre la misma utilizando como herramientas principales Python y el toolkit Intel©OpenVINO™. Los detalles de la aplicación se muestran en el capítulo 4.

4. Realizar las mediciones y hacer un resumen de los resultados obtenidos y las conclusiones que se extraen de ellos. Se muestran en el capítulo 5.

5. Finalizar desarrollando las conclusiones a las que se han llegado tras el trabajo realizado. Se exponen en el capítulo 6.

Durante todos los capítulos de esta memoria se exponen los problemas que han ido surgiendo en cada una de las fases y cómo se han ido solucionando o paliando. Por último, la memoria finaliza con el capítulo 7, donde se realizan una serie de reflexiones sobre el trabajo futuro en este campo.

Capítulo 2

Estado del arte

En este capítulo analizaremos el estado del arte de los diferentes conceptos y técnicas con los que hemos tratado durante el desarrollo de este trabajo.

2.1. Deep Learning

2.1.1. Definición

El término Deep Learning fue acuñado por primera vez por Detcher[2] en 1986. No obstante, previamente, en 1971 ya se había creado un algoritmo funcional que agrupaba hasta 8 capas de perceptrones[3] y en 1980 se había ideado un método de entrenamiento para redes neuronales profundas[4]. Es por eso que sabemos que es un concepto que se había comenzado a investigar desde mucho antes.

El Deep Learning es un subconjunto de técnicas de aprendizaje automático que se caracterizan por ser capaces de inferir características abstractas de los datos que se le proveen como entrada. No existe una sola definición de Deep Learning sino que, en general, todas cumplen que en mayor o en menor medida se asemejan a la organización del sistema nervioso humano. Esta disciplina busca que los algoritmos imiten la percepción humana y puedan tomar decisiones como si de estos últimos se trataran. Algunas de las aplicaciones más comunes en este campo reside en la visión por computador o el análisis y comprensión del habla humana.

En este trabajo nos centraremos en aplicaciones basadas en redes neuronales convolucionales, una de las técnicas de Deep Learning más desarrolladas para el análisis de imágenes en tiempo real.

2.1.2. Redes Neuronales Convolucionales

En Deep Learning, una de las técnicas más habituales son las redes neuronales convolucionales (CNN en adelante). Fueron inicialmente introducidas por Yann LeCun et al. en 1998[5] a través de la red que llamaron "Neocognitron" y que permitía que la propia red pudiera discriminar las características más importantes de una imagen independientemente de posibles cambios en su posición. Este tipo de redes suelen ser implementadas primero por varias capas de nodos PCA (análisis de componentes principales), seguidas de multitud de capas ocultas completamente conectadas entre ellas y una última capa de clasificación que se modifica para ajustar la red neuronal al problema que se esté abordando. Son redes en las que la convolución se erige como la operación por excelencia.

La convolución es una función matemática que permite ante dos funciones f y g, generar una tercera función que representa la superposición de f con respecto a una versión desplazada e invertida de g. Esta definición matemática se puede simplificar incluyendo la convolución como un tipo de media móvil, si se escoge una de las funciones iniciales como la función de característica de un intervalo.

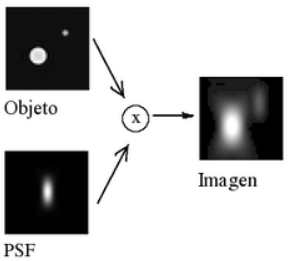


Figura 2.1: Convolución en un dispositivo óptico.

En visión por computador, este tipo de redes son ampliamente utilizadas pues nos permiten comparar imágenes y extrapolar características comunes a ambas. En la figura 2.1

podemos observar la convolución de dos imágenes, una operación básica en este tipo de redes. En el caso de una CNN, la imagen de entrada es operada junto a un núcleo previamente entrenado. Este núcleo es el que se genera a raíz del entrenamiento de la red neuronal y consiste en un "resumen" de las características que la red ha considerado como discriminantes.

La estructura de las CNN contiene además capas de neuronas de reducción de muestreo. Estas capas provocan que las neuronas más alejadas de la entrada sean mucho menos sensibles a cambios en los datos provistos. No obstante, son activadas por características mucho más complejas. Esto se puede observar ya en la estructura de LeNet introducida Yann LeCun et al. y mostrada en la figura 2.2.

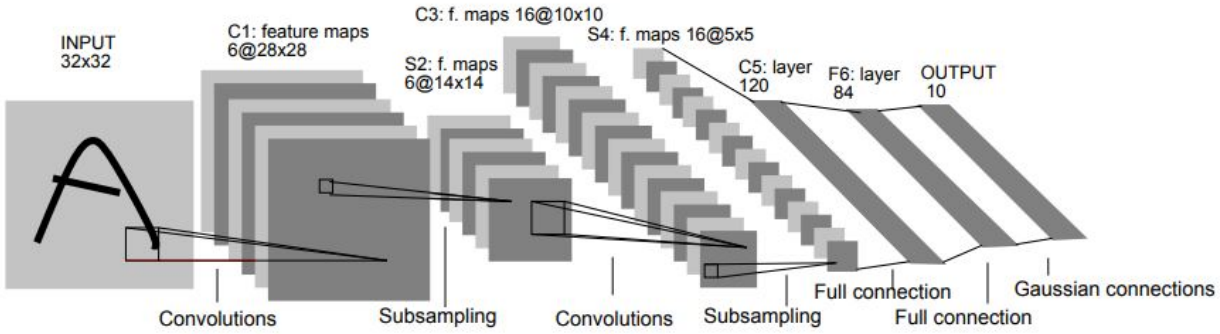


Figura 2.2: Estructura de LeNet propuesta por Yann LeCun et al. en "Gradient-based learning applied to document recognition".

Arquitectura Interna de una CNN

El funcionamiento de una red neuronal convolucional se basa fundamentalmente en la combinación de varias capas que realizan filtros sobre el conjunto de datos de entrada y propagan los resultados al resto de capas posteriores. En las CNN, gran parte de estas capas son de convolución.

Las capas convolucionales reciben una matriz que representa la imagen a clasificar (habitualmente en formato RGB). Con ella, la red neural divide la matriz original en otras

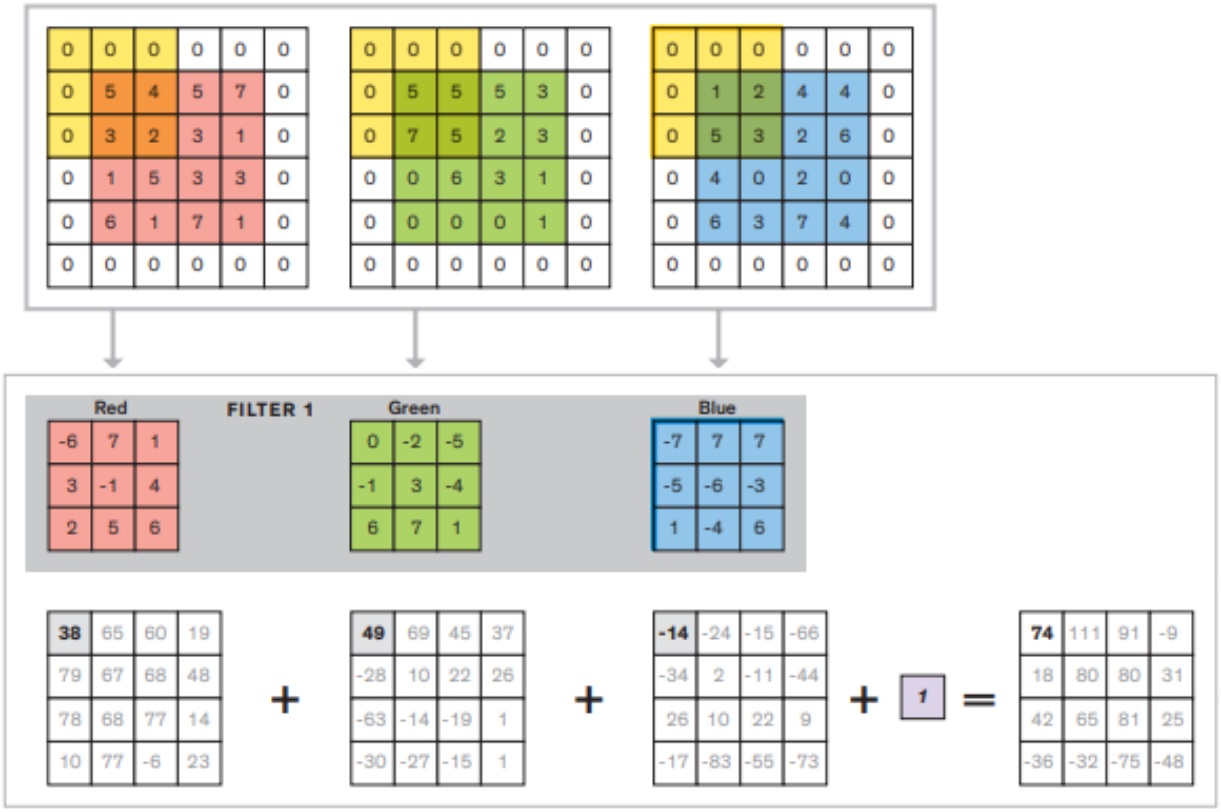


Figura 2.3: Gráfico del proceso de convolución dentro de una CNN.[6]

tres más pequeñas separadas por canales. Cada una de estas matrices será operada de forma individual. Los filtros se aplican sobre las matrices en bloques de NxN aplicando un producto a cada elemento de la matriz por cada elemento del filtro, destacando así características discriminantes de la imagen (gradiente de color, líneas rectas, etc). Cada nuevo elemento de la matriz resultado es el producto escalar de la matriz por el filtro (ej. $38 = 0 \times (-6) + 0 \times 7 + ... + 2 \times 6$). Cada nuevo bloque se obtiene desplazando el original un pixel a la derecha tras cada operación.

Tras aplicar el filtro sobre todos los posibles bloques de las matrices de cada canal, se obtienen tres matrices nuevas. Estas matrices se sumarán entre ellas y, opcionalmente, les será sumado también un valor constante denominado *bias*.

Al aplicar cada filtro, sumar y normalizar los resultados, se obtiene un conjunto de valores denominado *"feature map"* o mapa de caracterísiticas. Este proceso puede ser observado en la figura 2.3.

Finalmente, los resultados se propagan por todos los filtros de cada capa hasta llegar a una serie de capas totalmente conectadas entre ellas. En estas últimas, los mapas de características son examinados en su totalidad. La red neuronal finaliza con una última capa de clasificación en función de los valores obtenidos.

Un ejemplo de arquitectura interna de una CNN muy general se puede observar en la figura 2.4.

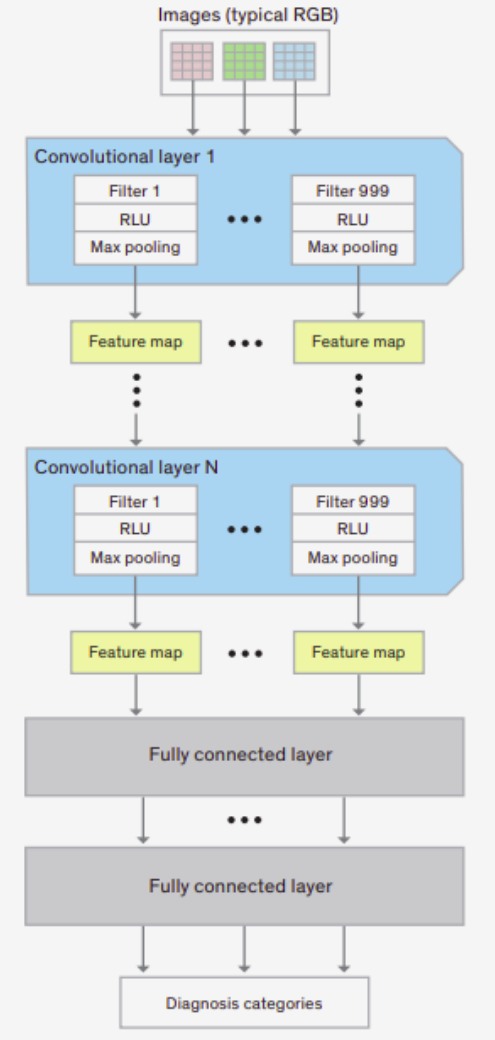


Figura 2.4: Arquitectura interna de una CNN genérica.[6]

Fase de Desarrollo de una CNN

Cuando hablamos de desarrollar e implementar una técnica de Deep Learning y, especialmente en aquellas basadas en aprendizaje supervisado como son las CNN, el primer paso consiste en obtener un conjunto de datos adecuado al problema que se vaya a abordar. Además, dichos datos deben ir etiquetados para poder entrenar la red neuronal adecuadamente. Sin un dataset ajustado al problema, la red neuronal dará muy malos resultados independientemente de cómo esté desarrollada.

Una vez preparado el conjunto de datos comienza el desarrollo de la red neuronal propiamente dicha. Este proceso se suele dividir en tres fases:

- Desarrollo: El desarrollo de una CNN comienza con la definición e implementación de la misma red. Esta fase es crucial y de alta complejidad dado que cuanto más esfuerzo

se realice en esta fase, mejores resultados obtendremos en las siguientes. Esta fase se subdivide comúnmente en otras dos más pequeñas:

  - Definición: La definición de una red neuronal consiste en el diseño de la misma. En esta fase es donde se toman las principales decisiones de diseño que van a condicionar el resto de las fases. Decidir el número de capas, las operaciones de los perceptrones (neuronas) y las conexiones entre los mismos resulta una tarea vital para el buen comportamiento de la red. En muchas ocasiones, dada la complejidad que esto supone en redes neuronales profundas, se opta por la adaptación de topologías públicas que suelen tener soporte por los principales frameworks de desarrollo.
  - Entrenamiento: Una vez implementada la topología de la red neuronal, llega el momento de asignar pesos a cada uno de los perceptrones. Durante el entrenamiento de la red neuronal, ésta última se limitará a ajustar dichos pesos para afinar los resultados obtenidos y "conducirlos" hacia los resultados esperados por el desarrollador. Para realizar el entrenamiento, se precisa de un dataset de elementos de entrada etiquetados con la clase a la que pertenece cada elemento. De esta forma, la red neuronal puede crear esos núcleos prototípicos de clase que se mencionaron en el apartado anterior. Una parte del dataset se reserva para el testeo y validación de la red neuronal.

- Despliegue: El despliegue de la red neuronal consiste en la instalación y puesta en marcha de la misma en el entorno de ejecución final. Se deben implementar funciones que transforman la información de entrada en "crudo" para adecuarla a la entrada esperada por la red. De la misma forma, se deben implementar funciones que analicen la salida y tomen decisiones en base a ella.

- Inferencia: La fase de inferencia es la fase final del desarrollo. Cuando la red neuronal está desplegada correctamente y comienza a funcionar. Una inferencia es una predic-

ción de clase sobre un elemento provisto en la entrada que no pertenece a ninguno de los dataset de entrenamiento, validación o testeo. Por ejemplo, si en una fotografía hay caras humanas o no. Resulta importante mencionar que, de forma general, la fase de entrenamiento utiliza operadores de mayor precisión (FP32) dada la necesidad de aplicar gradientes[7]. Mientras que en la fase de inferencia, se utilizan operadores más pequeños[8][9] (FP16, INT8) debido al ahorro computacional que suponen.

Esta división será importante más adelante para comprender el uso del toolkit Intel© OpenVINO™.

2.2. Reconocimiento Facial

El reconocimiento facial ha sido una de las mayores metas del aprendizaje automático en las últimas décadas. Su investigación ha derivado en multitud de conceptos y técnicas que se han podido aplicar a otros casos de uso y ha sido uno de los grandes motores de la visión por computador.

En este ámbito, es importante realizar una distinción clara entre el reconocimiento facial (FR) y la detección de caras (FD). En FD, el objetivo consiste en dilucidar si existen o no caras humanas dentro de una imagen y dónde se hallan dentro de la misma. Por contra, el FR consiste en saber si una cara determinada pertenece o no a un objetivo concreto.

2.2.1. Historia

Como mencionan M. Wang y W. Deng[10], el reconocimiento facial comienza con el uso de técnicas holísticas como el ya clásico algoritmo de EigenFaces[11]. Este algoritmo surge de la primera intuición de crear una representación de baja dimensionalidad de imágenes de caras humanas. Se pensó que el PCA (Análisis de Componentes Principales) podría extraer una serie de características comunes a cada cara, formando así una reconstrucción de la cara objetivo denominada eigenface. No obstante, las técnicas holísticas no conseguían reaccionar adecuadamente ante características que se salieran de las asunciones que se hubieran definido

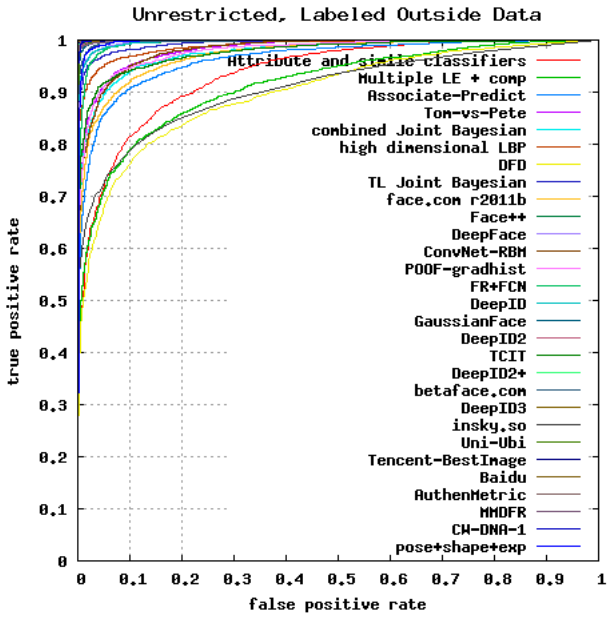


Figura 2.5: Curvas ROC medias sobre 10 ejecuciones de los mejores 19 algoritmos. Imagen extraída de la página de resultados de LFW[13].

previamente. Por ello, una componente de aprendizaje de caracterísitcas locales fue añadida, al que, con el paso de los años, se les fueron añadiendo mayor cantidad de filtros y parámetros de mayor dimensionalidad.

Finalmente, con la explosión de las redes neuronales y el Big Data, se han conseguido construir dataset públicos específicos para FR y eso ha motivado la creación de concursos públicos y benchmarks específicos. Estos hechos han derivado en un desarrollo exponencial de este tipo de técnicas que han finalizado con las redes neuronales convolucionales como el estándar de facto de la industria para realizar FR actualmente.

De entre los benchmarks más conocidos, destaca el benchmark "Labeled Faces in the Wild"(LFW)[12], del cual podemos extraer fácilmente una idea bastante aproximada del estado del arte en esta materia. Tal y como figura en su página web, las redes neuronales y los algoritmos desarrollados alcanzan precisiones casi comparables a la capacidad humana (muchas de ellas sobrepasando el 99.90 % de acierto).

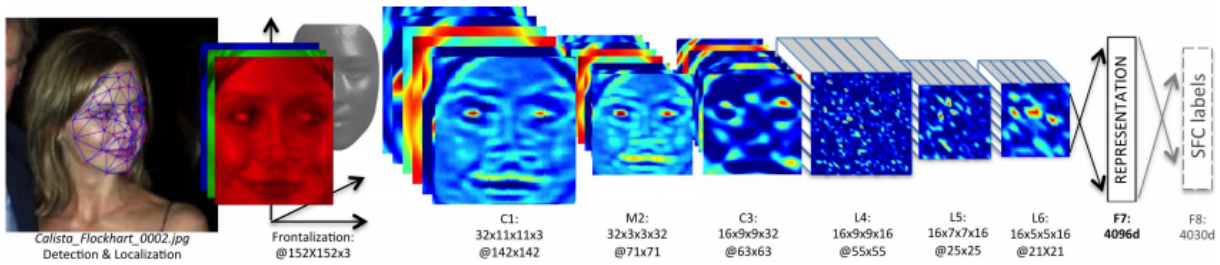


Figura 2.6: Architecture of DeepFace.

2.2.2. Modelos y técnicas

DeepFace

DeepFace[14] es uno de los modelos más precoces en cuanto a niveles de precisión casi humanos(97.35 ± 0.25). Creado por empleados de Facebook y publicado en 2014, no solo aporta porcentajes muy altos de predicción sino que además lo hace soportado por una red neuronal de muchas menos capas (9) que otros modelos del estado del arte del momento. Su topología está basada en una AlexNet como se puede comprobar en la figura 2.6. Por contra, necesita de grandes cantidades de datos para alcanzar estos niveles de precisión. El modelo original fue entrenado con un dataset propio de Facebook denominado Social Face Classification (SFC) de casi 4.4 millones de imágenes de 4030 personas distintas.

CoCo loss

CoCo loss[15] es actualmente uno de los métodos con mayor precisión del estado del arte (99.86 %). Se distingue de otros modelos en su método de entrenamiento, denominado *cosine loss*. Este método basa su efectividad en disminuir la distancia coseno, aumentando de esta forma la distancia entre clases y disminuyendo la varianza intra-clase. Además, no aplica un segundo entrenamiento sobre el conjunto de test.

Adicionalmente, el método COCO utiliza también la pose de la persona y de la cara para realizar la predicción. En la figura 2.7 se puede observar el algoritmo que sigue sobre una imagen concreta. En primer lugar, utiliza como base la posición de la cara de la persona. Después, realiza una detección del cuerpo al que pertene la cara y extrae una serie de *patches*

(zonas de interés recortadas de la imagen) que, junto a la cara y el cuerpo, se introducen en la red neuronal como parte de la entrada. En todo momento se realiza un *alignment* (transformación de la imagen a la posición referencia) de las imágenes.

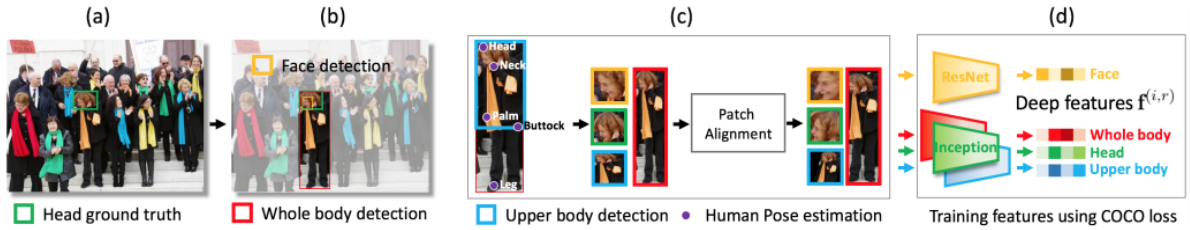


Figura 2.7: Algoritmo de CoCo loss.

SphereFace

SphereFace[16] es un modelo de reconocimiento facial que introduce la función A-Softmax loss (Angular Softmax), en la que se aplica un margen angular para distanciar las caracteristicas discriminantes de cada una de las clases. Está basada en una arquitectura ResNet[17] de 64 capas.

FaceNet

FaceNet es el modelo de reconocimiento facial en el que está basado este trabajo. Fue presentado en 2015 por F. Schroff, D. Kalenichenko, y J. Philbin[18]. Es un modelo que permite la representacion de las características faciales en un solo formato denominado *embedding*, compuesto por 128 valores. Como se puede observar en la figura 2.8, un embedding es un vector que resume las caracteristicas discriminantes de una cara humana determinada. Esta representación ha demostrado ser muy útil dado que nos permite realizar una comparación directa con otros embedding. En este caso, para conocer si dos embedding representan la misma cara, basta con calcular la distancia euclídea entre ambos. Si el resultado es menor que un umbral previamente fijado, podemos afirmar que es la misma cara. Este procedimiento evita la necesidad de aplicar técnicas de clasificación posteriores a la inferencia como *Support*

Vector Machines (SVM) o Análisis de Componentes Principales (PCA), metología bastante habitual en clasificación de imágenes mediante redes neuronales.

Este modelo presenta una precisión del 99.65 % según el benchmark LFW[12] y es ampliamente adoptado por la comunidad dada su simplicidad. Habitualmente es implementado utilizando Tensorflow[19] y varias versiones de Inception[20] como topología de red. En este trabajo se ha utilizado una topología Inception Resnet-v1. En la figura 2.8 se muestra un ejemplo real de aplicación de estos embeddings. Disponemos de 3 embeddings diferentes, dos de los cuales (B y C) pertenecen a la misma cara. El umbral ha sido empíricamente fijado en 1.2.

$$A = \begin{bmatrix} -0{,}0457619, 0{,}1293915, -0{,}0316607, \\ -0{,}0340939, -0{,}0560946, -0{,}0381384, \\ \dots \\ 0{,}1105223, 0{,}0640519, -0{,}0385667 \end{bmatrix}$$

$$B = \begin{bmatrix} -0{,}0239239, -0{,}0061291, -0{,}0948852, \\ -0{,}0508952, -0{,}0372205, -0{,}0342746, \\ \dots \\ -0{,}0131129, 0{,}1026618, 0{,}1148589 \end{bmatrix}$$

$$C = \begin{bmatrix} 0{,}0581797, 0{,}1263990, -0{,}1173248, \\ -0{,}1074057, -0{,}0122532, -0{,}1002343, \\ \dots \\ 0{,}0469320, 0{,}0187737, 0{,}0711396 \end{bmatrix}$$

$$d(A, B) = 1{,}475706$$

$$d(B, C) = 0{,}6011642$$

Figura 2.8: Embeddings de FaceNet inferidos de 3 imágenes distintas: A, B y C.

De forma general, se ha podido observar empíricamente que el número de caras a inferir es un elemento determinante del rendimiento, disminuyendo este último conforme aumenta el número de caras en la imagen. Con el objetivo de intentar mejorar este decremento en el rendimiento, se propone un clúster de aceleradores de la fase de inferencia.

2.3. Edge Computing

2.3.1. Definición

Según Cisco Internet Business Solutions Group[21], para 2020 habrá más de 50.000 millones de dispositivos conectados a Internet. Muchos de ellos serán sensores, turbinas, motores, cámaras de vigilancia, etc. La utilidad de estos dispositivos radica mucho más en la disponibilidad de los datos que en su utilidad. Ahora bien, con la nueva tendencia IoT, se le está dando mucha importancia a que dichos elementos, generadores de datos, estén lo más cerca posible de los dispositivos que los procesan y, por ende, les aporten valor.

Algunas aproximaciones actuales consisten en crear pequeños centros de procesamiento de datos que permitan situarse mucho más cerca de la demanda de procesamiento. De esta forma, se consiguen reducir las latencias y mejorar la disponibilidad de los datos procesados. A esta aproximación se le llama computación en la frontera o Edge Computing. De acuerdo con W.Shi[22], el Edge Computing es definido como *"the enabling technologies allowing computation to be performed at the edge of the network, on downstream data on behalf of cloud services and upstream data on behalf of IoT services"*.

No obstante, en ocasiones, existe la necesidad de que la respuesta del sistema sea completamente en tiempo real, por ejemplo, en un coche autónomo. Es en estos casos donde, ni siquiera un centro de datos localizado a 50 kilómetros, puede ser suficiente para responder a tiempo a cualquier eventualidad. Es por eso que precisamos dispositivos que, además de ser generadores de información, también sean procesadores de los mismos. Esta aproximación permitiría que una cámara de seguridad pueda reconocer de inmediato la cara de una persona que va dentro de un coche a muy alta velocidad o que un coche pueda activar los frenos de emergencia y modificar la dirección en base a lo que percibe para evitar un accidente repentino.

Este aspecto del Edge Computing es el que se investiga actualmente junto con los métodos de Deep Learning para crear una nueva clase de dispositivos inteligentes que se puedan instalar en los generadores de datos (o junto a ellos), de modo que la respuesta a cualquier

evento sea inmediata. Además se busca que tomen decisiones óptimas que además puedan ser modificadas en respuesta a cualquier variación de la situación observada. Adicionalmente, cuando transladamos esta cuestión a la frontera (*edge*) encontramos muchas restricciones de latencia y ancho de banda. Al realizar la inferencia de los datos en los propios dispositivos IoT podemos reservar dichos recursos limitados para tareas más pesadas pero con menos prioridad.

Es por esta razón que grandes empresas de la industria del hardware como Nvidia o Intel han decidido realizar grandes inversiones en hardware específico que permita la aceleración de las redes neuronales en dispositivos IoT para su despliegue en la frontera.

2.3.2. Tecnologías de aceleración de la fase de inferencia

Como se ha mencionado en la sección anterior, actualmente existen muchas alternativas en el mercado para conseguir acelerar redes neuronales en la frontera. En este trabajo destacamos las alternativas de Nvidia e Intel como las más asentadas en la industria y la de Google, que al momento de redactar este documento acaba de finalizar su período de beta.

Tecnologías Intel©

Intel OpenVINO[23] es un toolkit, creado y mantenido por Intel, de optimización y despliegue de soluciones de Deep Learning para su uso en IoT e Inteligencia Artificial. Sus principales características son las siguientes:

- Optimización de modelos de redes neuronales profundas. Uno de sus módulos, el Model Optimizer, es el encargado de optimizar y comprimir el modelo ya entrenado para que su uso en inferencia y su despliegue sea óptimo.

- Soporta los principales frameworks de desarrollo de redes neuronales: Tensorflow, Caffe, MXNet, OMMX. También soporta las principales topologías de red: YOLO, Inception, Resnet, MobileNet, etc. Esto favorece que los desarrolladores puedan implementar la red en el lenguaje y el framework que prefieran para más tarde optimizarlo con

OpenVINO.

- Permite optimizar los modelos utilizando técnicas determinadas y con diferentes precisiones de punto flotante (FP32, FP16, INT8). Esto permite generar un modelo óptimo con el objetivo de ahorrar recursos para la inferencia.

- El módulo Inference Engine permite programar fácilmente la inferencia de los datos. Este módulo además soporta distintos tipos de plug-in para abstraer la implementación de la inferencia del dispositivo físico en el que se va a desplegar.

- Pone a disposición del usuario módulos específicos para Intel CPU, Intel GPU, FPGA, VPU, MYRIAD, HETERO (plugin heterogéneo) y MULTI.

- Los plugins anteriormente mencionados abstraen al programador de la capa física en la que se despliega el modelo. Tienen soporte y balanceo de carga integrado para multi-dispositivo y soportan multi-inferencia sobre un mismo dispositivo.

- El Inference Engine dispone de una API nativa en C++ tanto síncrona como asíncrona y una API en Python que consiste en un *wrap* de la API en C++.

- El framework completo es compatible con Windows, Ubuntu y MacOS. Además, el Inference Engine puede ser instalado de forma individual sobre Raspbian.

- El plugin MYRIAD soporta el Intel Movidius Neural Compute Stick 2 (MYRIAD X) y tiene retrocompatibilidad con el Intel Neural Compute Stick (MYRIAD 2).

Gracias a todas estas funcionalidades, OpenVINO ha demostrado ser un framework extremadamente útil para el despliegue de servicios inteligentes en IoT. En la figura 2.9 podemos observar la estructura y comunicación entre los principales componentes del toolkit.

Junto con OpenVINO, Intel ha desarrollado los Intel Movidius Neural Compute Stick[24] y su revisión, el Intel Neural Compute Stick 2[25]. Ambos son dispositivos USB que contienen

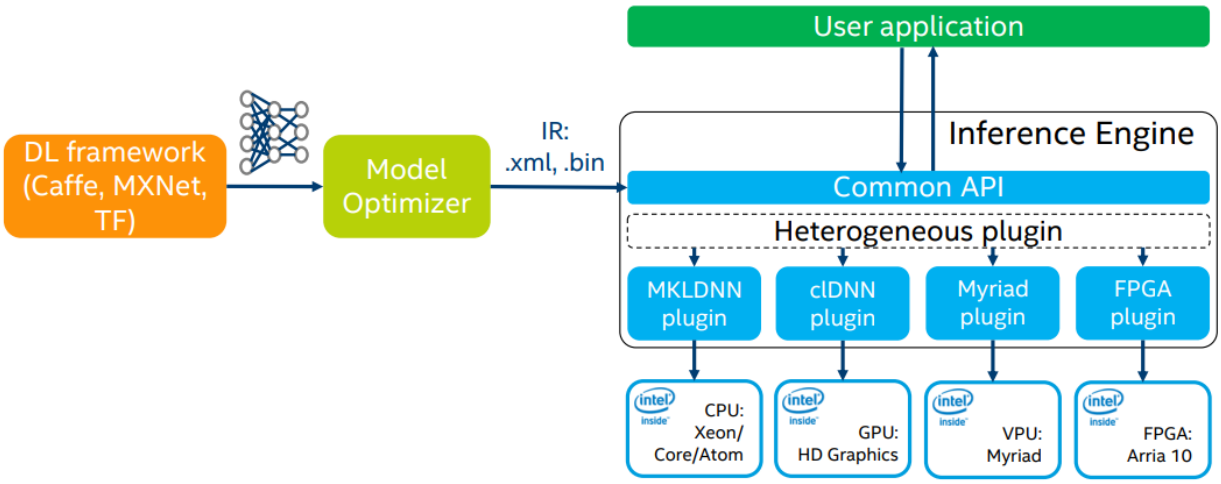


Figura 2.9: Estructura y comunicación de los componentes de OpenVINO.

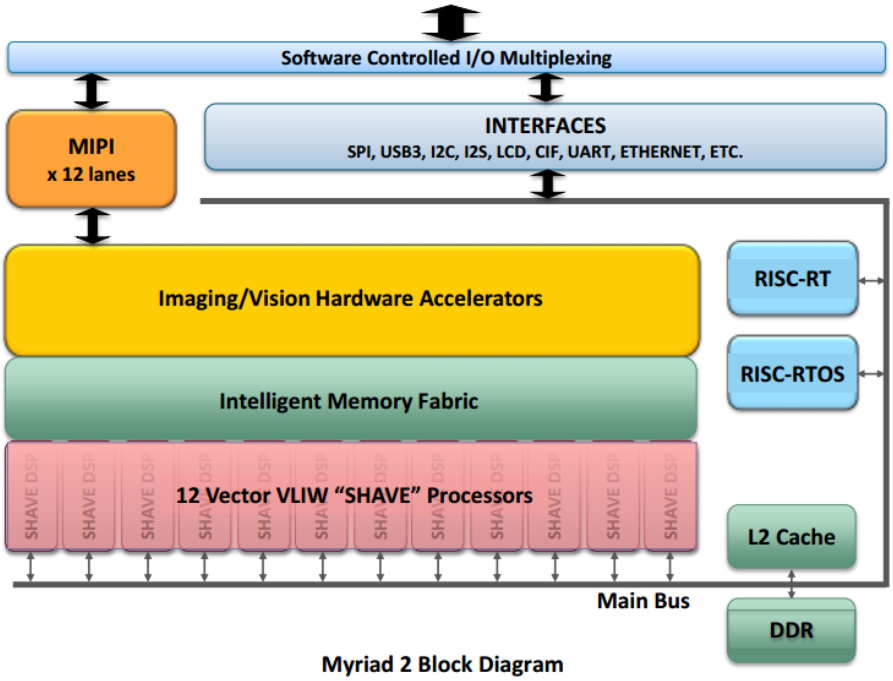


Figura 2.10: Estructura interna del acelerador MYRIAD 2, incluido en los dispositivos Intel Movidius Neural Stick originales.

un procesador vectorial para la acelaración de las inferencias: el MYRIAD 2 y MYRIAD X respectivamente.

Como podemos ver en la figura 2.10, el chip basa su rendimiento en 12 vectores de procesamiento VLIW (Very Long Instruction Word) denominados SHAVE. Los procesadores VLIW se distinguen por utilizar un tipo de paralelismo a nivel de instrucción. El número de instrucciones es muy reducido, pero estas son de gran longitud. Esto es así debido a que en la misma instrucción se define el estado de todas las unidades funcionales del sistema dejando toda la responsabilidad de planificación al compilador (o al programador si programa en ensamblador). Esto es lo contrario de lo que se realiza en los procesadores superescalares en los que es el hardware el que planifica de forma dinámica las instrucciones. El uso de instrucciones VLIW permite simplificar el hardware provocando un descenso radical del consumo de recursos físicos y lógicos. Pese a que los VLIW son procesadores poco utilizados para el uso general, son idóneos para sistemas IoT ya que se busca la minimización del consumo.

Entre el Myriad 2 y el Myriad X hay pocas diferencias. Destaca especialmente el aumento de estos vectores SHAVE VLIW de 12 a 16.

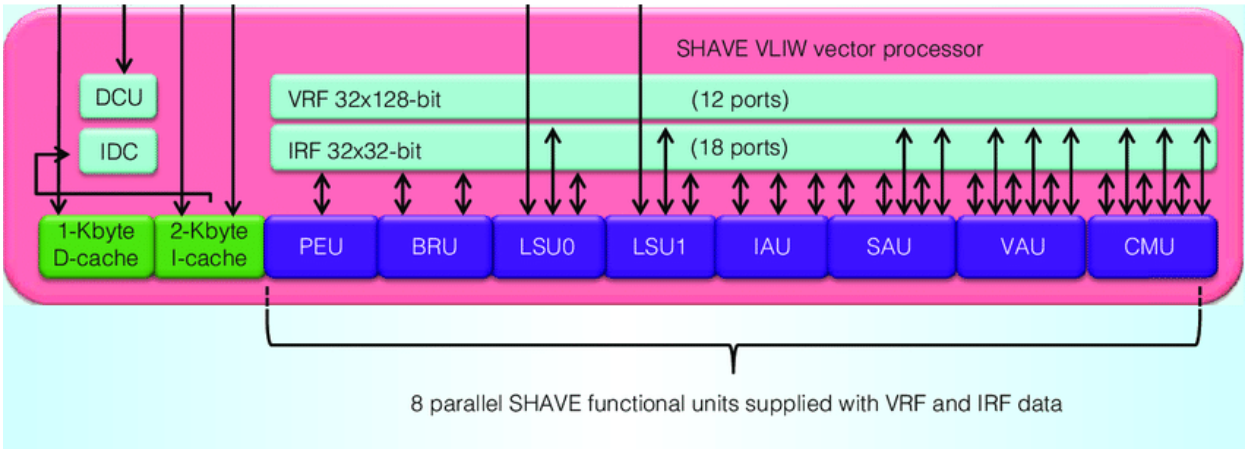


Figura 2.11: Estructura del acelerador VLIW SHAVE 3.0 creado por Movidius e introducido en los chip MYRIAD 2.

Tecnologías de Nvidia©

TensorRT es el equivalente de OpenVINO de Nvidia. Al igual que el anterior, TensorRT se centra en la optimización de redes neuronales profundas para su uso en inferencia. Permite desplegar sus soluciones en data centers distribuidos, la nube o en sistemas empotrados. Basa su funcionamiento en CUDA, el lenguaje de programación paralela de Nvidia.

Dispone de las siguientes características:

- Soporte para los principales frameworks de desarrollo (Tensorflow, Caffe, OMMX...).
- Optimizador de modelos para inferencia.
- Compilación con diferentes niveles de precisión en punto flotante (FP32, FP16).
- Soporte para despliegue sobre GPU's Tesla, sistemas Jetson y plataformas NVIDIA Drive.

Además, TensorRT pone a disposición del desarrollador un TensorRT Inference Server que permite el despliegue de soluciones en datacenters, implementando microservicios encapsulados en contenedores. Está basado en Docker y Kubernetes para su máxima integración con estrategías DevOps.

Junto a TensorRT como framework de desarrollo, Nvidia ha desarrollado toda una familia de dispositivos empotrados para el despliegue de aplicaciones de inteligencia artificial. Se trata de la familia Nvidia Jetson. Esta familia consiste en una línea de miniordenadores muy potentes y optimizados al máximo para acelerar inferencias.

En general, todos son dispositivos de muy bajo consumo (5-10W para la Jetson Nano) pero con especificaciones bastante potentes. Cada producto de la línea tiene bastantes diferencias con respecto a los demás, pero en general todos se ajustan a las siguientes características:

- Arquitectura GPU NVIDIA (Maxwell, Pascal, Volta) con 128-512 núcleos CUDA y hasta 64 núcleos Tensor.

- Memoria eMMC 5.1.
- Memoria RAM LPDDR4 o LPDDR4x.
- Codificación de vídeo a 4K.
- Procesadores ARM de 4 a 8 núcleos.
- Cámara integrada.

Como podemos observar, son especificaciones muy potentes si pensamos en que son dispositivos dedicados al mundo IoT y sobretodo teniendo en cuenta el consumo tan reducido que presentan.

Google Coral Toolkit

La alternativa de Google para este mercado es muy reciente. Google Coral es un toolkit dedicado al desarrollo de aplicaciones de inteligencia artificial (fundamentalmente redes neuronales) enfocado en el despliegue de estas últimas sobre dispositivos IoT. En este sentido, ofrece características y funcionalidades muy similares a las mencionadas en TensorRT y OpenVINO.

Adicionalmente, y al igual que sus competidores, Google también ofrece hardware específico para este tipo de aplicaciones. Por un lado, ofrece un acelerador USB compatible con placas de uso común como las Raspberry Pi y por otro, una placa de desarrollo donde crear pruebas de concepto y experimentos. Ambos dispositivos se basan en Edge TPU (Unidades de Procesamientos basados en tensores) y Tensorflow Lite para los modelos precompilados. En caso de querer integrar aplicaciones de Machine Learning o Inteligencia Artificial en sistemas legacy, Google ofrece una serie de dispositivos más específicos en varios formatos de forma distintos. Por último, es importante mencionar que Google Coral tiene compatibilidad con Google Cloud para poder aprovechar los servicios de Cloud TPU de Google.

En favor de una comparación más precisa, es necesario mencionar que la placa de desarrollo de Google Coral se asemeja más a una placa de bajo coste como la Raspberry Pi al

disponer de 1GB de memoria LPDDR4 y 8GB de memoria eMMC que a las placas de la familia Jetson, cuyo rendimiento es muy superior. Igualmente, esta placa de desarrollo no está pensada para ser desplegada en entornos de "producción", sino que para ello dispone de las placas PCIe con diferentes factores de forma mencionadas anteriormente. Por último, los aceleradores USB de Google Coral contienen un ASIC diseñado por Google para la inferencia de modelos de Machine Learning, permitiendo la ejecución de MobileNetv2 a 400 FPS[26]. La composición de este ASIC no ha trascendido más allá de lo especificado anteriormente.

En la tabla 2.1 podemos observar una pequeña comparación de estas placas.

Placa	Precio($)	Consumo (Watt)	Especificaciones
Google Coral DEV Board	149.99	NA	NXP i.MX 8M SoC (Quad-core Cortex-A53, plus Cortex-M4F) 1 GB LPDDR4, 6 GB eMMC Edge TPU coprocessor Cryptographic coprocessor
Nvidia Jetson Nano	109.00	15	NVIDIA 128-core Maxwell Dual-Core NVIDIA Denver 2 64-Bit CPU Quad-core ARM A57 4GB LPDDR4, 16GB eMMC 5.1
Raspberry Pi 3B+	48,42	5	Broadcom BCM2837B0 Cortex-A53 SoC@1.4GHz 1GB LPDDR2 SDRAM
Raspberry Pi 4	66,04	9	Broadcom BCM2711 Cortex-A72 1,5GHz Quad-Core Broadcom VideoCore VI 500MHz 4GB LPDDR4-2400 SDRAM

Tabla 2.1: Comparación de las placas mencionadas del estado del arte.

Capítulo 3

Arquitectura

En este capítulo desarrollaremos la arquitectura tanto hardware como software utilizada para el estudio. El objetivo es implementar y desarrollar una aplicación de reconomiento facial basada en Facenet. Una vez realizada la implementación, aceleraremos las inferencias utilizando Intel OpenVINO Toolkit y varios Intel Neural Stick 2. Por último, desplegaremos la solución sobre distintos host y mediremos su rendimiento y potencia (esto último a través de un sensor INA260).

3.1. Hardware

La arquitectura hardware del sistema se muestra en la figura 3.1. Con el objetivo de mejorar la aceleración de la aplicación se propone el siguiente clúster[27] formado por varios NCS2 que son conectados a un hub activo USB 3.0. Este elemento servirá para asegurar que llega la energía necesaria a los aceleradores. El HUB estará conectado a uno de los puertos USB del Host que, en este estudio, comprenden un portátil con procesador i7 de 4ª generación, una Raspberry Pi 3B+ o una Raspberri Pi 4. Adicionalmente, se ha utilizado un sensor de corriente y potencia INA260 situado entre la toma de corriente, el host y el HUB USB 3.0. En este último caso el objetivo era comprobar la escalabilidad del consumo con respecto al aumento de NCS2 y el incremento que esto suponen sobre un dispositivo empotrado de las características de la Raspberry Pi. Los valores de corriente y potencia que mide el INA260 son capturados, a través de la interfaz I2C del mismo, por una Raspberry

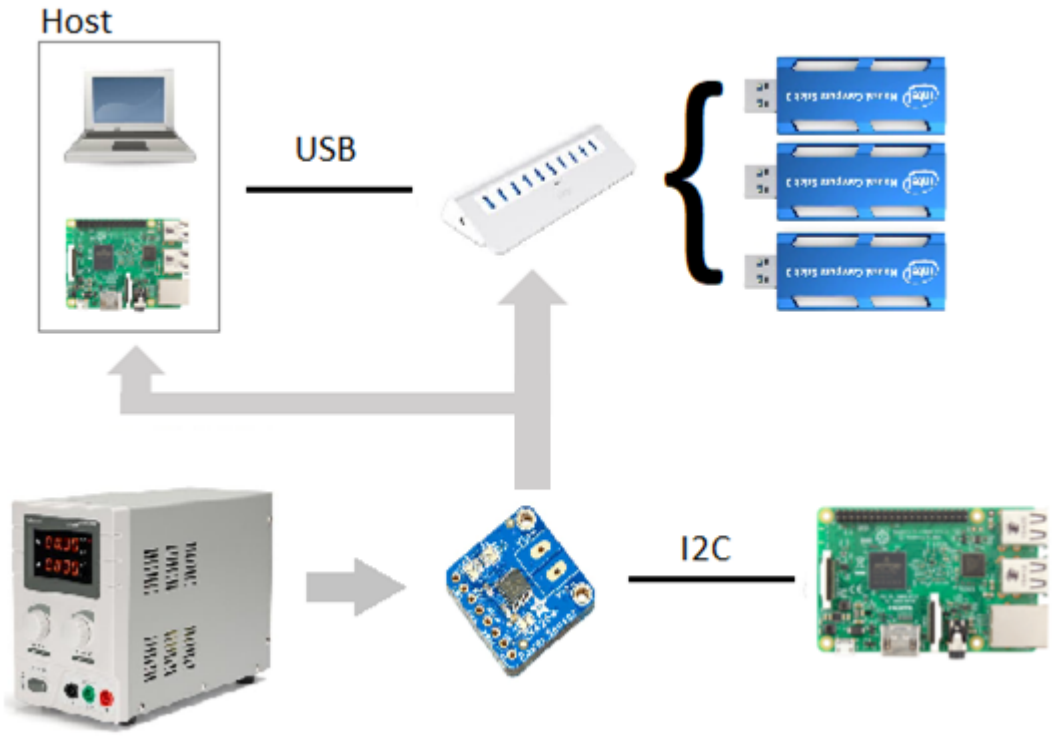


Figura 3.1: Arquitectura Hardware de la Aplicación.

Pi 3B+ adicional que se encuentra fuera del circuito de la aplicación y solo es utilizada como dispositivo medidor independiente. A continuación, detallamos cada uno de los elementos junto con los problemas que se han ido presentando y las soluciones propuestas.

3.1.1. Intel Neural Compute Stick 2

Como ya hemos mencionado, utilizaremos los NCS2 de Intel como herramienta de bajo coste para acelerar las inferencias de los modelos utilizados. Esto nos aporta varias ventajas, entre ellas el bajo consumo, coste de mercado y facilidad de adaptación al tratarse de una interfaz USB 3.0. De la misma forma, se han tenido que tomar ciertas medidas para resolver los problemas que presenta crear un clúster de estos dispositivos:

- Alimentación de los NCS2: Se ha podido observar que un host de bajo consumo como una Raspberry Pi 3B+ no es capaz de alimentar más de 2 NCS2 simultáneamente provocando un consecuente descenso del rendimiento debido a la pérdida de información

por una conexión inestable. Por esta razón, se ha utilizado un hub USB 3.0 activo que provee de la energía necesaria a los aceleradores.

- USB 3.0: Pese a ser una de sus mayores virtudes, se presenta un problema a la hora de usarlos sobre placas IoT de propósito general como la Raspberry Pi 3B+ dado que esta última no dispone de USB 3.0, provocando un evidente cuello de botella a la transmisión de datos. Afortunadamente, a mediados de este año 2019, salió al mercado la esperada Raspberry Pi 4 que sí tiene soporte para USB 3.0 y que fue añadida a este estudio para completarlo.

- Balanceo de Carga: Los NCS originales eran soportados por una herramienta llamada NCSDK que permitía realizar un balanceo de carga artesanal por parte del programador y daba a este último control casi total sobre el hardware que utilizaba. Un estudio de escalabilidad con esta herramienta y varios NCS ya fue llevado a cabo por Panadero. T[28]. Lamentablemente, los NCS2 solo son compatibles con el toolkit OpenVINO que no disponía de esta opción hasta la tercera revisión de 2019 y que aislaba al programador del planificador. Este hecho condicionó la aplicación durante gran parte del estudio.

Con todo, en este estudio se han utilizado 3 NCS2 para tomar medidas de rendimiento y consumo. Se ha analizado sobre todo el escalado del rendimiento, pues uno de los objetivos prioritarios de este estudio era analizar la viabilidad de despliegues aprovechando el escalado horizontal de este tipo de dispositivos.

3.1.2. Host

Con el objetivo de hacer una comparativa completa en cuanto a rendimiento bruto, escalabilidad y consumo; se han realizado pruebas con diferentes dispositivos host. De esta forma, evidenciamos mejor el efecto que pueda tener el acelerador (en este caso el NCS2), sobre el rendimiento y consumo de los dispositivos. En este estudio, se han utilizado tres dispositivos distintos, todos de uso general:

- Un portátil que contiene un Intel Core i7-4702MQ@2.20GHz, 8GB RAM DDR3 y 500GB SSD SATA3. A nivel de sistema operativo utiliza Ubuntu 16.04.06-LTS.
- Una Raspberry Pi 3B+ que contiene un SoC Broadcom BCM2837B0 quad-core A53 (ARMv8) 64-bit@1.4GHz, 1GB LPDDR2 SDRAM y una Micro-SD de 32GB. A nivel de sistema operativo utiliza Raspbian 9.
- Una Raspberry Pi 4 que contiene un SoC Broadcom BCM2711 Quad CoreCortex A-72@1.5GHz, 4GB SDRAM LPDDR4-2400Mhz y una Micro-SD de 32GB. A nivel de sistema operativo utiliza Raspbian 10.

Es importante mencionar que el toolkit OpenVINO utilizado solo dispone de soporte para CPUs Intel, por lo que solo se ha podido medir el rendimiento de la CPU en *standalone* en el portátil. Debido a esta razón, no hemos incluido dichos valores en la comparativa, aunque sí han sido recogidos.

3.1.3. INA260

El INA260 es un sensor de corriente, voltaje y potencia eléctrica compatible con lógicas de 3 y 5V. Es capaz de medir hasta 36VDC y 15A en ambos sentidos. Gracias a su interfaz I2C es compatible tanto con dispositivos Arduino como con placas de carácter general como las Raspberry utilizadas en este estudio.

Inicialmente, se pretendía utilizar el INA219, pero en anteriores estudios[28] se había evidenciado que un sistema parecido superaba los 3.2A de medición máxima. Para evitar tener que adquirir dos de estos chips y modificar sus direcciones de memoria como se proponía en el estudio, se planteó la adquisición del INA260 que dispone de una mayor capacidad de medición.

El modelo de conexión concreto que se ha utilizado para realizar las mediciones del INA260 sobre la Raspberry Pi es el mostrado en la figura 3.2. Con el fin de medir adicionalmente el consumo de la Raspberry, se ha utilizado el mismo modelo como se observa en la figura 3.3, donde se puede ver una imagen real del sistema ya montado.

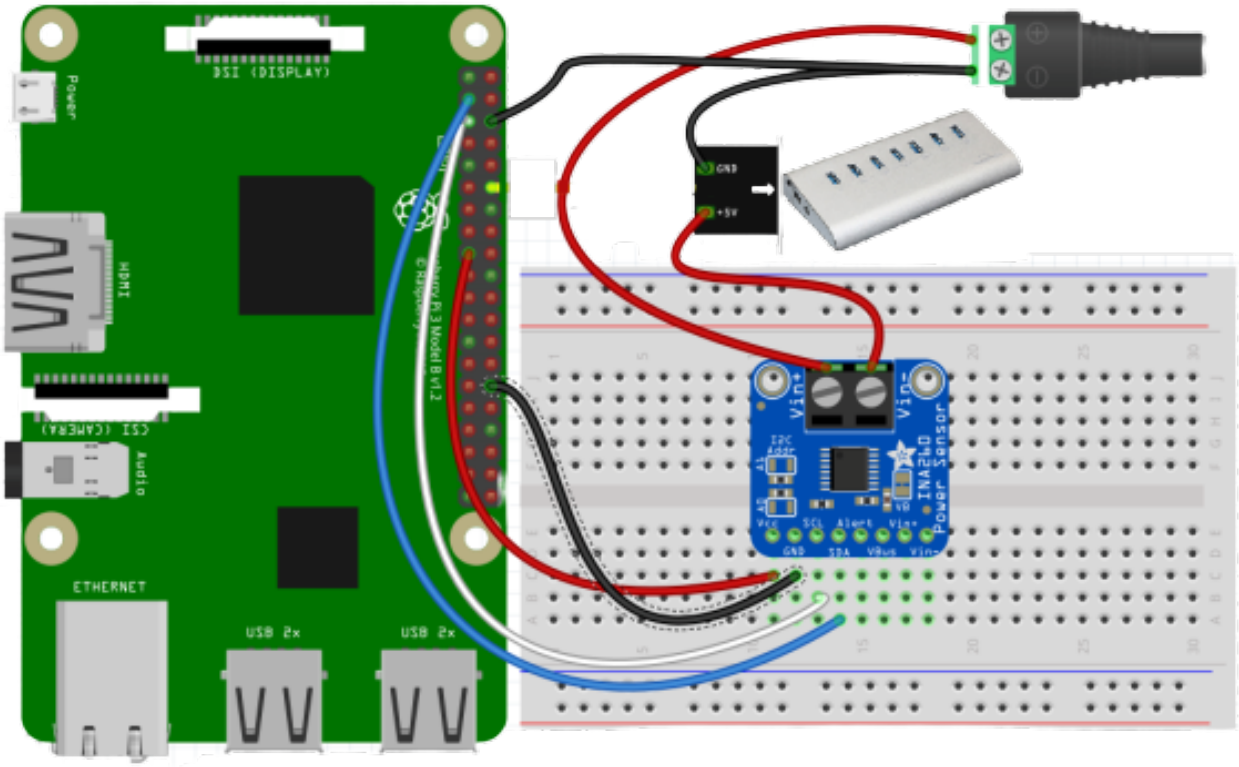


Figura 3.2: Conexión del INA260 al hub usb activo donde se conectarán los NCS2 y a los pines GPIO de la Raspberry Pi 3B+ que actuará como dispositivo lector de las mediciones.

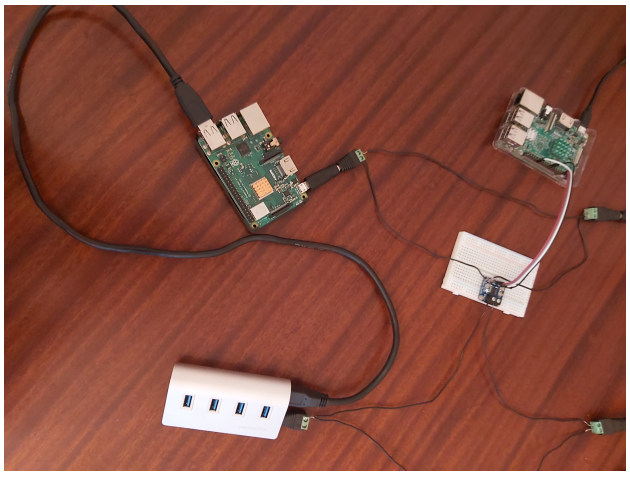


Figura 3.3: Fotografía de la arquitectura hardware momentos previos a la experimentación.

3.2. Software

3.2.1. Intel OpenVINO Toolkit

El toolkit Intel OpenVINO es la herramienta que se eligió para desarrollar la aplicación. Con él, el desarrollador tiene completa libertad para implementar y entrenar la red neuronal utilizando cualquiera de los framework de desarrollo soportados. Este grado de libertad permite poder adaptar modelos ya desarrollados y disminuye la curva de adopción del toolkit. En este estudio, se han utilizado dos de sus herramientas: el Model Optimizer y el Inference Engine.

- **Model Optimizer** (MO): Una vez entrenado el modelo, se ejecuta el MO sobre él. Esta herramienta permite ajustar multitud de parámetros para su optimización como el tamaño de batch, precisión, etc. Según el framework identificado por el MO o el indicado por el usuario, OpenVINO desarrollará una serie de transformaciones a la red neuronal para optimizar y comprimir la topología de cara a la fase de inferencia. La salida del MO consiste en dos ficheros IR (Intermediate Representation): un .xml para representar la topología de la red optimizada y un .bin que contiene los pesos de los perceptrones.

  Para este estudio, se ha utilizado una CNN para la detección facial (face-detection-retail-004) publicada por Intel y optimizada para ser ejecutada junto a OpenVINO. Para el reconocimiento facial, se ha recurrido a la implementación en Tensorflow de FaceNet desarrollada por David Sandberg y accesible desde su respositorio en GitHub[29]. La precisión media del modelo de detección facial es del 83 %, entrenado y probado sobre el dataset público WIDER[30]. Dado que este dataset contiene caras siempre mayores o iguales a tamaños de 60x60 pixeles, no se puede asegurar resultados equivalentes en caras de menor tamaño. Ambos modelos han sido optimizados usando precision FP16.

- **Inference Engine** (IE): Los dos ficheros IR generados se utilizan en el IE para crear

un objeto Executable Network con el que seremos capaces de realizar la inferencia. Este objeto contiene además mucha información como las dimensiones de los parámetros de entrada y salida, el número de capas, etc. Cada Executable Network, si se hace uso de la API asíncrona, puede soportar múltiples peticiones de inferencia, las cuales se ejecutarán paralelamente. El programador podrá acceder en cualquier momento al estado de cualquiera de las peticiones y recoger el resultado cuando de verdad lo necesite. El objetivo fundamental del IE es ejecutar las inferencias de la manera más transparente posible para el usuario. Por esta razón, el IE selecciona automáticamente el plugin a utilizar en función de las preferencias del usuario y de los dispositivos compatibles disponibles en ese momento. A continuación balancea las inferencias asignando cada una al dispositivo con menos carga de trabajo.

También cabe mencionar que, aunque la abstracción de la capa física permite que los programadores no tengan que hacer programas específicos para cada dispositivo, también niega la capacidad de optimizar el balanceo de las inferencias o de monitorizar el estado de los mismas, perdiendo así información valiosa en entornos críticos.

Esta circunstancia se ha ido suavizando durante las últimas revisiones del toolkit. En estos momentos se pueden extraer algunos contadores de rendimiento, se ha introducido una herramienta de benchmarking y se pueden cargar modelos concretos en dispositivos físicos concretos. Este último punto fue de suma importancia para el desarrollo de la aplicación como veremos en capítulos posteriores.

3.3. Arquitectura global de la aplicación

En particular, para la aplicación desarrollada, el flujo software a seguir sería el mostrado en la figura 3.4. Los ficheros .pb de los modelos entrenados son optimizados utilizados el MO en el entorno de desarrollo. A continuación, el IE deberá ser instalado en el entorno de despliegue (ej, Raspbian solo dispone de compatibilidad con el IE al ser un SO orientado a dispositivos IoT). Los ficheros IR son cargados en el IE quien se encargará de inicializar

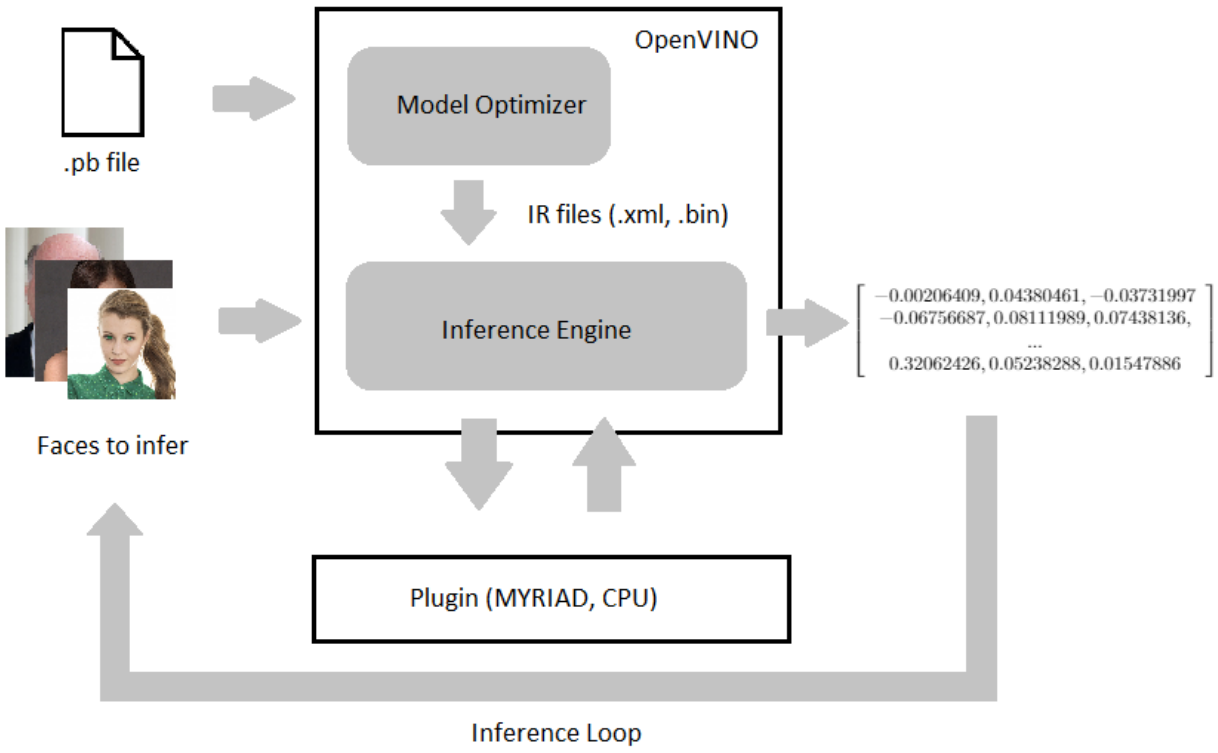


Figura 3.4: Arquitectura Software.

todas las estructuras de datos para realizar la inferencia sobre el dispositivo que se le haya indicado. Usando Python, se procesarán las imágenes de las caras. Estas imágenes, serán enviadas al IE para que sean inferidas y devuelva el embedding correspondiente a cada una.

Por la parte del hardware, el IE usará los ficheros del modelo para inicializar el plugin indicado (en este caso el Plugin MYRIAD) quién será el encargado de planificar y distribuir las inferencias sobre cada uno de los dispositivos MYRIAD disponibles. Cada NCS2 contiene un MYRIAD X que a su vez distribuirá los cálculos entre los 16 núcleos SHAVE de los que dispone. Este comportamiento se puede observar en la figura 3.5.

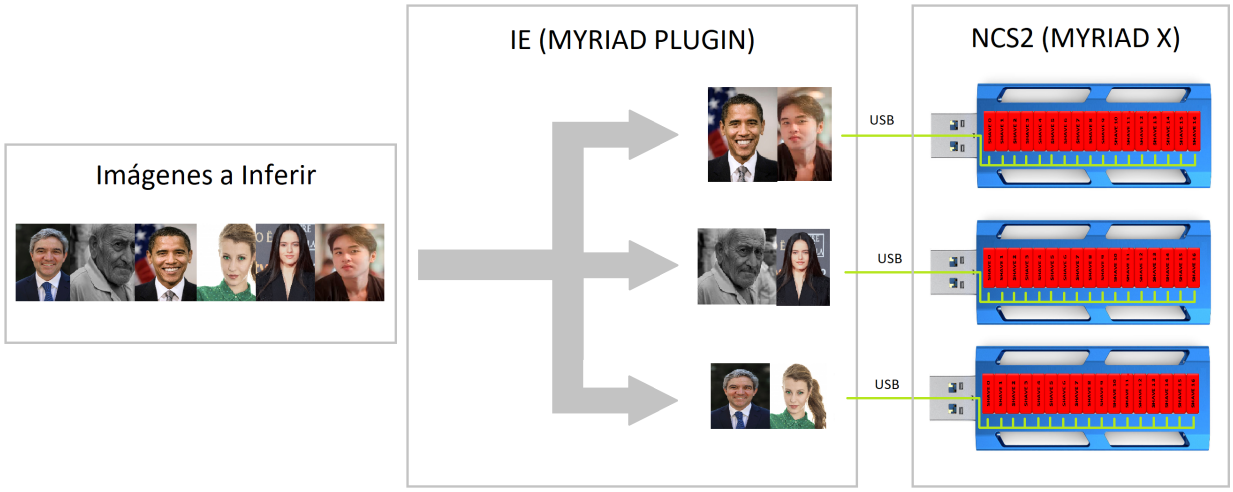


Figura 3.5: Distribución de las imágenes sobre los NCS2 y los SHAVE del MYRIAD X.

Capítulo 4

Aplicación

En este capítulo se detallará como se ha llevado a cabo la implementación de una aplicación sobre la arquitectura descrita anteriormente. La aplicación, por tanto, debe soportar la inferencia de imágen y vídeo y distribuirla sobre un clúster de NCS2 con el objetivo de reconocer una cara objetivo sobre dicha imagen o vídeo. Igualmente, la aplicación debe registrar estadísticas de control y rendimiento que nos permitan analizar y extraer las conclusiones sobre nuestros objetivos.

La aplicación ha sido desarollada utilizando el toolkit OpenVINO y Python 3.6. Adicionalmente, se han aprovechado las funcionalidades de OpenVINO que permiten la ejecución de inferencias asíncronas con el fin de mejorar el rendimiento global. El código del programa se encuentra accesible desde un repositorio de GitHub.

Como se ha explicado anteriormente, la aplicación recibe una imagen de una cara objetivo y un vídeo sobre el que buscar dicha cara. En primer lugar, inferirá la cara objetivo y a continuación comenzará a extraer los frames del vídeo. En cada frame del vídeo, el programa detectará las caras presentes y utilizará FaceNet para obtener sus correspondientes embeddings. Con ambos embedding, el de la cara objetivo y el de la cara candidata, se aplica la distancia euclídea para discernir si se trata de la misma persona o no. Finalmente, el programa muestra el vídeo inferido con las caras detectadas resaltadas y con un color determinado dependiendo de si dicha cara se trata o no de la objetivo. A todo este proceso lo denominaremos *pipeline* de inferencia y se ve representado a continuación en el algoritmo

1.

Algorithm 1 Pseudocódigo del pipeline de inferencia.

```
frame = getFrameFromVideo(input)
frame = preprocessFrameForDetection(frame)
faces = detectFaces(cnn, frame)
if len(faces) > 0 then
    for all face in faces do
        faceImage = cropFace(face, frame)
        faceImage = preprocessFaceForRecognition(faceImage)
        embedding = recognizeFace(facenet, faceImage)
        if euclidDistance(embedding, targetEmbedding) < THRESHOLD then
            markFaceRecognized(face, frame, true)
        else
            markFaceRecognized(face, frame, false)
        end if
    end for
end if
showResults(frame)
```

La estructura anteriormente descrita da pie a una implementación paralela siguiendo un modelo productor/consumidor y distinguiendo 3 procesos principales:

1. Captura de frames: El procesamiento del vídeo y de la cara objetivo se puede realizar de forma independiente del resto del programa. En este trabajo se procesa el vídeo, y si hay frames disponibles, estos son almacenados en una cola. Se trata de un proceso puramente productor.

2. Inferencia: Este trabajo resulta aún más sensible a la paralelización dado que se evidencia una mejora evidente al volver a paralelizar según el número de dispositivos de inferencia. En esta fase del pipeline, el programa inicializa dos objetos que contienen los dos modelos de redes neuronales utilizados: el *face_detector* y el *face_recognizer*. Se toman los frames producidos anteriomente y se aplica la detección de caras. Una vez detectadas, se lanzan inferencias asíncronas según el número de caras y el número de dispositivos. Cuando están disponibles los resultados (en forma de embedding),

se comparan una a una con la cara objetivo y se marcan en consecuencia. Por último, los frames procesados se meten en otra cola para ser mostrados. Es un trabajo consumidor/productor.

3. Muestra de resultados: Muestra por pantalla los frames procesados y calcula varias métricas de ejecución y rendimiento. Siempre es la última fase en terminar. Es un trabajo puramente consumidor.

Es importante indicar que para tomar las medidas de rendimiento propias de la inferencia (tiempos de inferencia, número de inferencias, etc), se aislan las mediciones al bucle de inferencia. No obstante, el cálculo de los FPS se realiza en la muestra de resultados. A la hora de tomar las medidas de rendimiento, se comentaron las partes que mostraban los frames procesados por pantalla para focalizar el esfuerzo en el bucle de inferencia.

Con todo ello, para implementar este pipeline se han tomado dos aproximaciones de ejecución paralela: multihilo y multiproceso.

4.1. Aproximación Multihilo

En esta aproximación, la aplicación crea un proceso para realizar las inferencias. A continuación, se crea un hilo de ejecución por cada NCS2 utilizado. Cada uno de los hilos solo se centra en el bucle de inferencia: recibe un frame, detecta las caras, aplica reconocimiento facial sobre ellas y guarda el frame procesado en una cola para que sea mostrado. Dado que dentro de un mismo proceso las referencias a los dispositivos físicos pueden ser compartidas, esta aproximación permite que dentro de un mismo frame las caras puedan ser repartidas entre todos los dispositivos de inferencia disponibles.

En el algoritmo 2 podemos observar el psecódigo de esta aproximación donde creamos un proceso de lectura de frames, obtenemos el número de dispositivos disponibles y por cada dispositivo lanzamos un hilo (thread) de ejecución del bucle de inferencia. Por último creamos un proceso de lectura de resultados. Es importante puntualizar que las líneas 2 a

la 6 se ejecutan en otro proceso aparte del programa principal.

Algorithm 2 Pseudocódigo del modelo multihilo

```
1: createInputReaderProcess()
2: numDevices = getNumDevices()
3: for i in range(0, numDevices) do
4:     thread = createInferenceThread(i)
5:     launchThread(thread)
6: end for
7: createProcessingResultsProcess()
```

En principio, este método debería ser menos demandante a nivel de recursos y, por tanto, la solución idónea de cara a un despliegue en dispositivos IoT de bajo consumo. No obstante, Python implementa un cerrojo conocido como GIL[31] (Global Interpreter Locker). Esto es debido a que el core de Python no es *thread-safe* (consistente ante la paralelización) por lo que el GIL se encarga de secuencializar los thread que se ejecuten desde el intérprete de Python.

4.2. Aproximación Multiproceso

Este modelo de ejecución es parecido al anterior con la salvedad de que ahora, en vez de hilos, inicializamos todo un proceso completo para cada uno de los dispositivos de inferencia. Este método no permite la compartición de los dispositivos físicos por lo que estos deben ser específicados manualmente en la inicialización del Inference Engine. En caso contrario, podrían darse casos en que el primer proceso aloje el *face_ detector* en un dispositivo y el *face_ recognizer* en otro distinto (el distribuidor de carga de OpenVINO siempre apunta al dispositivo más liberado). Esto provocaría que el siguiente proceso no pudiera crear ningún objeto de inferencia y, por tanto, produjera una excepción y finalizara la ejecución abruptamente. También existirían casos en el que el segundo proceso exisitiría su primer objeto de inferencia antes que el primer proceso alojara su segundo objeto, provocando que no hubiera colisión alguna y el programa se ejecutara con normalidad.

En la figura 4.1 se puede observar cómo el orden de inialización de cada uno de los objetos

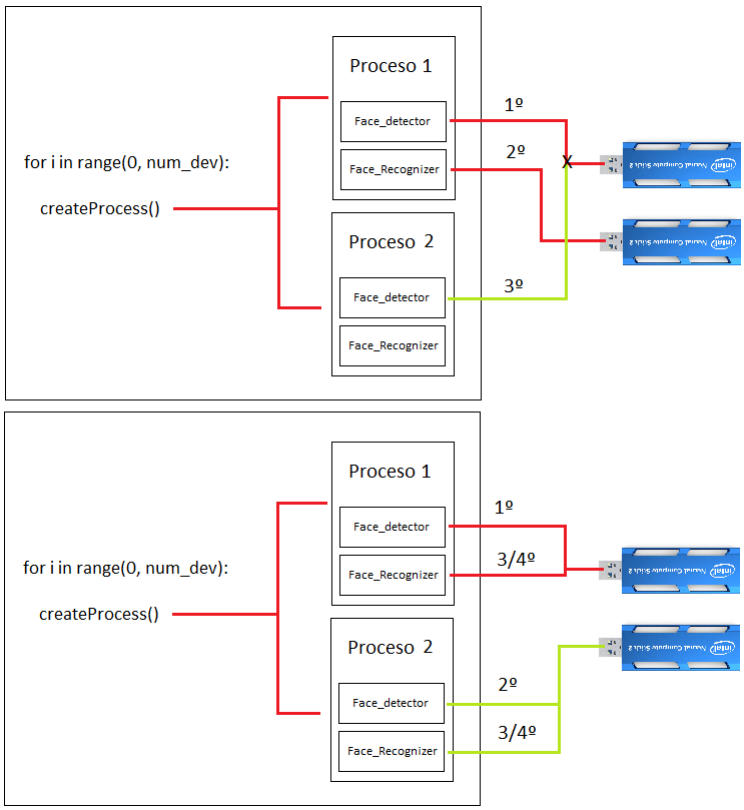


Figura 4.1: Ejecución de los procesos con colisión sobre el dispositivo físico y sin colisión dependiendo del orden de ejecución de las inicializaciones de los objetos de inferencia.

de inferencia puede provocar colisiones sobre los dispositivos físicos siguiendo la política de balanceo de carga de OpenVINO. En el primer caso, el primer proceso inicializa ambos objetos antes de que el segundo procesa pueda hacerlo, por lo que reserva los dos NCS2 disponibles y se provocan colisiones. En el segundo caso, la inicialización es alternativa por lo que cada proceso dispone de un NCS2 y, por tanto, no hay problemas de colisión.

En el algoritmo 3 podemos observar un pseudocódigo que representa la implementación llevada a cabo por esta aproximación. Al igual que en el anterior, se crean dos procesos auxiliares (de lectura de datos y muestra de resultados); pero en este caso los procedimientos de inferencia son creados como procesos diferenciados unos de otros.

Este modelo, pese a ser más voraz a nivel de recursos del sistema, se presupone como más potente porque otorga más recursos a cada uno de los procesos de inferencia. Además,

Algorithm 3 Pseudocódigo de la aproximación multiproceso.

```
createInputReaderProcess()
devices = getInferenceDevices()
for all device in devices do
    createInferenceProcess(device)
end for
createProcessingResultsProcess()
```

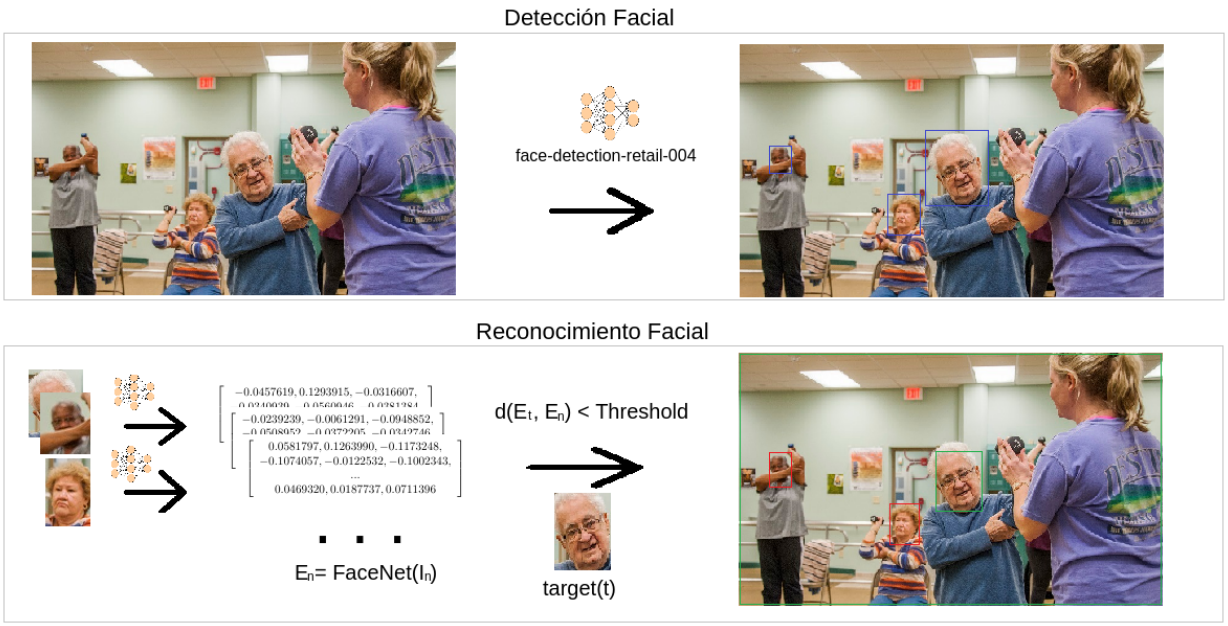


Figura 4.2: Ejemplo gráfico del funcionamiento de la aplicación sobre una imagen.

en este caso el GIL no aplica por lo que no tenemos un cuello de botella en la ejecución del paralelismo.

Para asegurar restricciones de tiempo real, en ambos modelos sólo se consume un frame si alguno de los procesos o hilos ya han finalizado el procesamiento y pueden inferir el nuevo frame de forma inmediata.

A continuación, se muestra en la figura 4.2 un ejemplo claro del funcionamiento de la aplicación en un ámbito hospitalario. Como se puede observar, la aplicación recibe una imagen que puede ser tanto una fotografía como el frame de un vídeo que está siendo tomado en tiempo real. Esta imagen es pasada a la red face-detection-retail-004 la cual es la encargada de realizar la fase de detección. En ella, la red neuronal detectará si hay

caras humanas en la imagen y facilitará las coordenadas donde las ha ido encontrando. La aplicación las marcará y serán facilitadas a la siguiente fase del pipeline.

En la fase de reconocimiento facial, iteramos sobre cada una de las caras encontradas, extrayendolas de la imagen para minimizar el ruido y aplicando FaceNet sobre ellas. Cada una de las caras dará como resultado un embedding de 128 valores. Por último, cada uno de los embedding será comparado con la cara objetivo mediante la distancia euclídea. Si esta distancia es menor que el umbral (threshold) fijado al comienzo de la aplicación, esta última entenderá que se trata de la misma persona y lo marcará en consecuencia dentro de la imagen original.

Capítulo 5

Experimentación

En este capítulo se detallará la experimentación realizada sobre la arquitectura descrita anteriormente en el capítulo 3 usando la aplicación detallada en el capítulo 4. Para asegurar la máxima rigurosidad de las mediciones, la velocidad del procesador se ha fijado al máximo manualmente (*performance mode*) y las conexiones USB se han monitorizado para asegurar que no exista un cuello de botella en la comunicación y transimisión de los datos.

Para evaluar el rendimiento, varios vídeos se han utilizado para realizar las pruebas. Estos vídeos son redimensionados a 300x300 píxeles para ser pasados a la primera red neural. Las siguientes medidas se toman durante la ejecución del programa: mínimo, máximo y tiempo de inferencia medio; el número medio de caras en relación con los FPS; el total de tiempo de ejecución de todo el programa y el número de inferencias calculadas. Además, se han tomado medidas de potencia y consumo utilizando el INA260 detallado en el capítulo 3.

5.1. Resultados en host de propósito general

En el primer experimento, se utilizó un portátil con un Intel Core i7 y un clúster de hasta 3 aceleradores NCS2. Además, los vídeos utilizados en el experimento han sido escogidos según el número de caras presentes a lo largo de todo el vídeo. De esta forma, la escalabilidad no solo se analiza a nivel hardware en cuanto al número de aceleradores, sino también a nivel software en función del número de caras. Los vídeos son de 1, 2, 5, 9 y 15 caras respectivamente y un caso especial de caras aleatorias etiquetado como R. Este último caso

introduce un número variable de caras a lo largo del vídeo. Los vídeos han sido seleccionados manualmente debido a la falta de un dataset específico que asegurara las restricciones que se habían fijado.

En primer lugar, se tomaron unas medidas base que sirvieran como control para los siguientes experimentos. Estas mediciones se realizaron ejecutando la aplicación sobre la CPU del portátil, sin ningún tipo de acelerador. Los resultados se pueden observar en la tabla 5.1. Es claramente visible la disminución del rendimiento conforme aumenta el número de caras llegando a casos peores de apenas un frame por segundo con 15 caras.

Número de caras	Min FPS	Avg FPS	Max FPS
1	15	21.84	26
5	1	6.03	8
15	1	1.78	7
R	1	9.77	14

Tabla 5.1: Rendimiento de FaceNet sobre un Intel Core i7-4702MQ@2.2GHz. Los frames por segundo son tomados sobre vídeos de 300x300 píxeles.

Después de esto, se realizaron las pruebas con el modelo multihilo. En la figura 5.1, podemos observar los resultados obtenidos de dichas pruebas. Como se podía esperar, el descenso del rendimiento en función del número de caras se mantiene con respecto al caso anterior. Adicionalmente, se puede apreciar una diferencia significativa con respecto al rendimiento sin aceleradores. Por otro lado, apenas se puede apreciar un speedup significativo con el aumento del número de NCS2 . Esto sugería la existencia de un cuello de botella en cuanto al rendimiento. Con el objeto de aislar la causa de este hecho, se han medido los tiempos de inferencia medios en función del número de NCS2. Los resultados de esta medición figuran en la tabla 5.2.

Como se puede observar, las diferencias presentes en todos los casos son mínimas y no dan lugar a sospechar que exista una limitación a nivel hardware de los aceleradores. Esto derivó a buscar la causa en el software tanto de la aplicación como del propio toolkit.

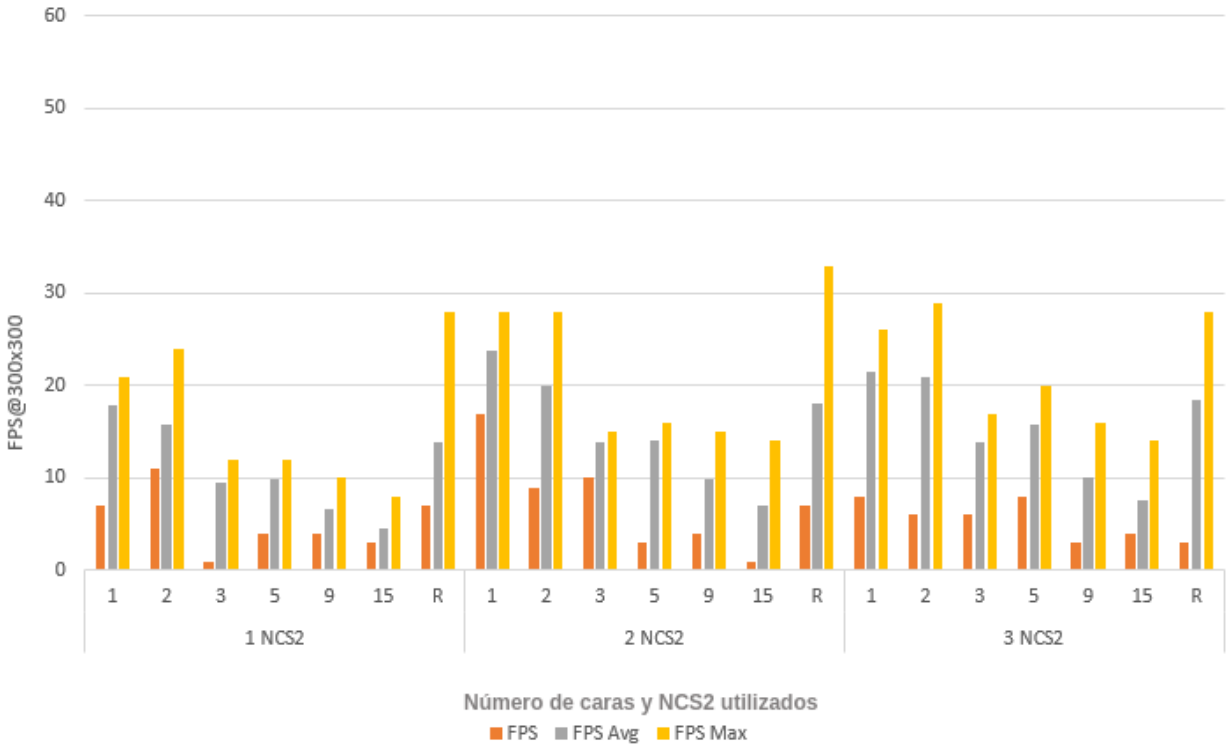


Figura 5.1: Resultados obtenidos de la ejecución del modelo multihilo.

Para comprobar si el problema podía residir en el balanceador de carga de OpenVINO se procedió a realizar los experimentos utilizando el modelo de multiproceso. Es necesario recordar que dicho balanceador de carga basa su funcionamiento en alojar las peticiones de inferencia en el dispositivo que tenga menos carga de trabajo. Los resultados de este segundo experimento se pueden ver en la figura 5.2.

Como muestra la figura, en el modelo de multiproceso sí existe un escalado directo según el número de NCS2. Además, se alcanzan resultados bastante altos, consiguiendo por encima de 10 FPS@300x300 en el caso peor del test con número de caras aleatorias. Esto refuerza la hipótesis de partida ya que podemos observar que un mayor número de aceleradores sí provoca una mejora en todos los casos.

Para afianzar estos resultados, se han calculado los speedup efectivos de este experimento. Como podemos observar en la tabla 5.3, los speedup se encuentran muy cerca de los valores ideales de 2 y 3 para 2 y 3 NCS.

1 NCS2	Número total de inferencias: 2279 Mínimo: 14.17 ms Medio: 20.16 ms Máximo: 25.41 ms
2 NCS2	Número total de inferencias: 2279 Mínimo: 14.17 ms Medio: 19.91 ms Máximo: 24.78 ms
3 NCS2	Número total de inferencias: 2279 Mínimo: 14.22 ms Medio: 19.84 ms Máximo: 25.90 ms

Tabla 5.2: Tiempos de inferencia medios en el caso de caras aleatorias usando el modelo de multihilo con 1,2 y 3 NCS2.

Estos resultados tan prometedores del modelo multiproceso chocan directamente con los resultados obtenidos del modelo mulithilo. Dado que ambos utilizan la misma política de balanceo de carga, podemos descartar que esta sea la razón del no escalado del primer modelo. Esto reduce las posibilidades, confirmando al GIL como la causa con mayor probabilidad de provocar este comportamiento.

A continuación, se trazó la relación a lo largo del tiempo de ejecución entre los FPS alcanzados y el número de cara medios en cada segundo. Con esta representación, resulta mucho más claro que el aumento computacional derivado del incremento de caras se ve claramente suavizado a nivel de rendimiento conforme añadimos más NCS2. es destacable que el caso peor con 3 NCS2 no baja de 25 FPS@300x300. Esto corrobora que el uso de los NCS2 no solo aumenta el rendimiento pico de la aplicación sino también la resiliencia al aumento de carga. Estas gráficas se pueden observar en la figura 5.3.

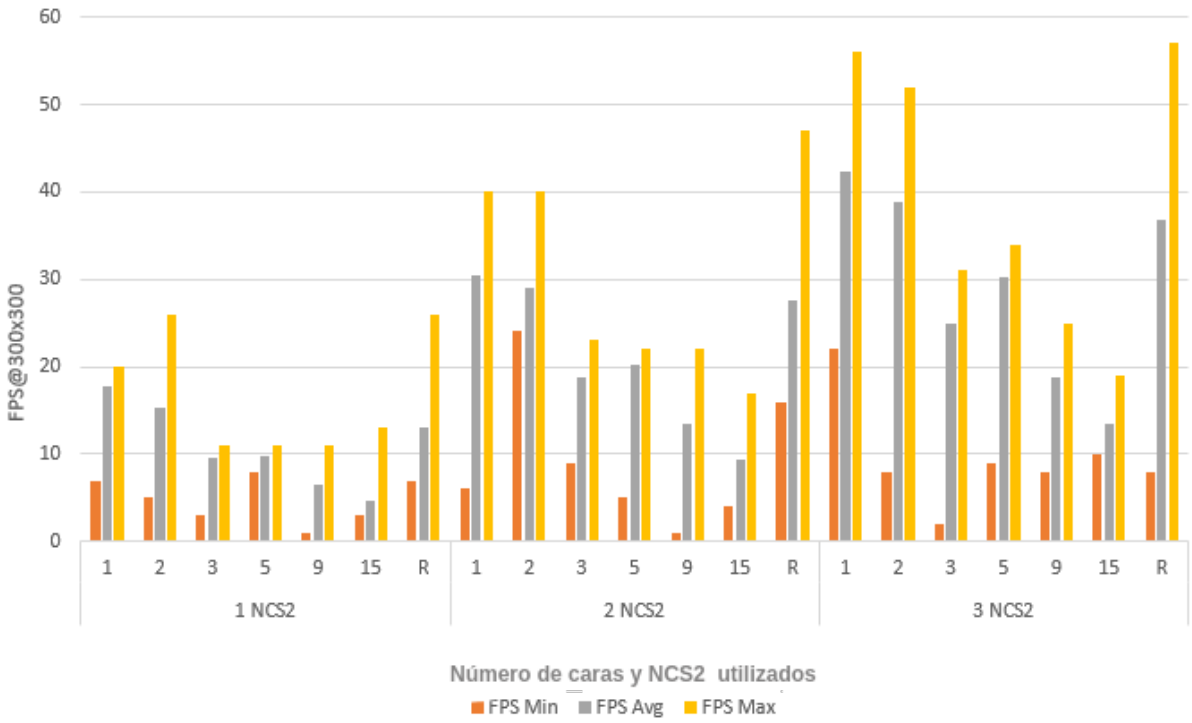
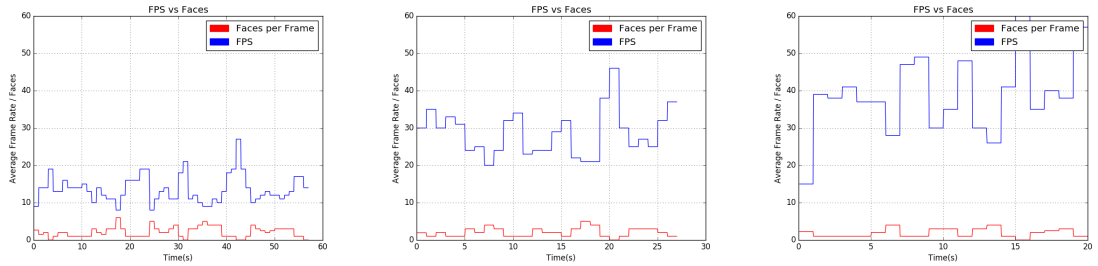


Figura 5.2: Resultados obtenidos de la ejecución del modelo multiproceso.

N. NCS2	Caras	FPS	Speedup
2 NCS2	1	30.42	1.71
	2	29.16	1.89
	3	18.87	1.98
	5	20.16	2.00
	9	13.53	2.00
	15	9.38	1.99
	R	27.60	2.00
3 NCS2	1	42.40	2.38
	2	38.76	2.51
	3	25.00	2.63
	5	30.16	3.07
	9	18.87	2.91
	15	13.50	2.87
	R	36.80	2.80

Tabla 5.3: FPS@300x300 medios y speedup empleando la aproximación multiproceso con respecto al número de caras por vídeo y el número de NCS2 utilizados para acelerar la inferencia.



(a) 1 NCS2 (b) 2 NCS2 (c) 3 NCS2

Figura 5.3: Relación entre el número medio de caras y los frames procesador por segundo.

5.2. Resultados sobre plataformas empotradas

En este apartado se desarrollan los resultados obtenidos de un experimento similar al anterior pero utilizando como hosts sobre una Raspberry Pi 3B+ y una Raspberry Pi 4. En vista de los resultados obtenidos en el apartado anterior, en este experimento sólo se han tomado las medidas con el modelo de multiproceso con el fin de hacer una comparativa más justa entre todas las plataformas.

5.2.1. Resultados en Raspberry Pi 3B+ como host de la aplicación

En la figura 5.4 se muestran los resultados obtenidos de la ejecución del experimento utilizando una Raspberry Pi 3B+ como host de la aplicación. Es fácilmente observable que hay un sensible descenso del rendimiento en comparación con los resultados obtenidos en el portátil. Por otro lado, sí es visible una mejora entre el caso monodispositivo (un solo acelerador NCS2) y multidispositivo (varios NCS2) pero apenas es apreciable un pequeño speedup a la hora de introducir más dispositivos en el segundo caso (de 2 NCS2 a 3 NCS2). Este comportamiento puede ser debido a la falta de interfaces USB 3.0 que introducen un cuello de botella inevitable al rendimiento. Con el objetivo de comprobar esta hipótesis y al mismo tiempo intentar mejorar el rendimiento del sistema se ha procedido a realizar el experimento sobre una Raspberry Pi 4 cuyos resultados son comentados en la siguiente sección.

Pese a todo ello, es necesario puntualizar que la Raspberry Pi 3B+ supone una dismi-

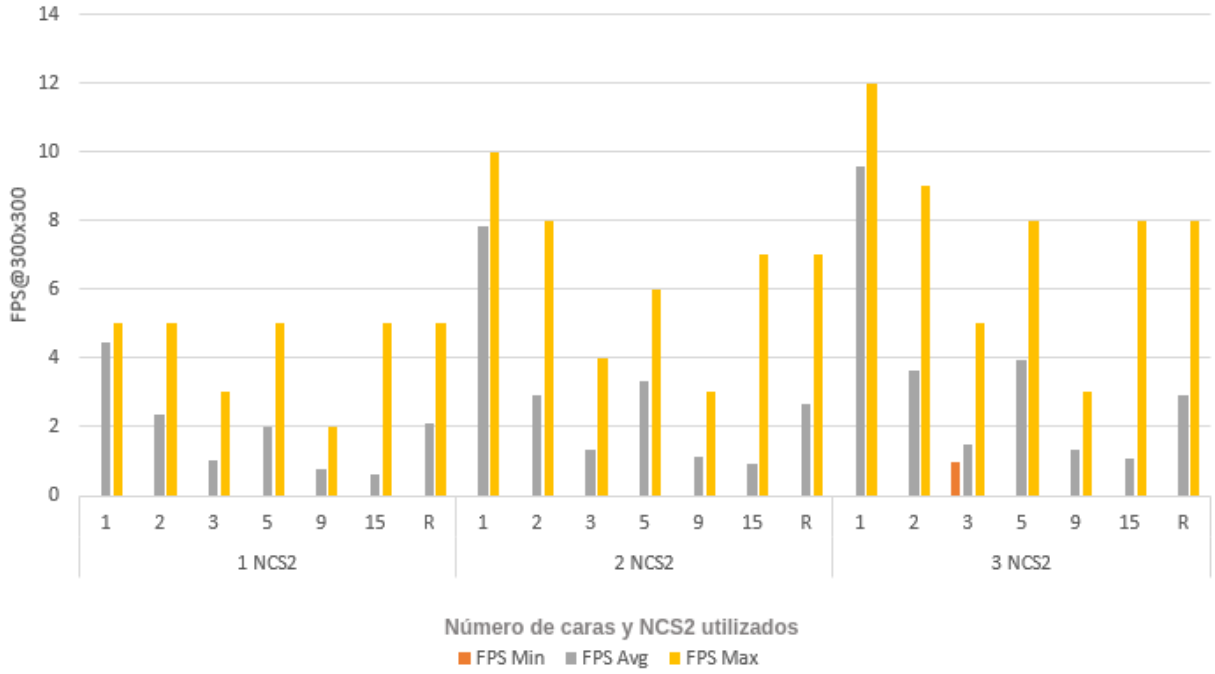


Figura 5.4: Rendimiento del modelo multiproceso variando el número de NCS2 y usando una Raspberry PI 3B+ como host de la aplicación.

nución radical del consumo por lo que una comparación directa con el portátil como host no sería acertada para los objetivos propuestos.

Para afianzar esta última afirmación, se procedió a tomar muestras de la potencia consumida por la arquitectura utilizada junto a la Raspberry Pi 3B+. Para ello se utilizó el INA260 siguiendo el modelo planteado en el capítulo 3. Las medidas se han tomado para el clúster de la Raspberry Pi 3B+ en sus distintas configuraciones: con 1, 2 y 3 NCS2. De esta forma se puede medir el consumo total del sistema y, al mismo tiempo, el incremento que supone un solo acelerador sobre el consumo general.

Los resultados de estas mediciones se muestran a continuación en la figura 5.5. Como se puede ver, se nota claramente el inicio y el final del período de inferencia, el cual provoca una fluctuación muy rápida de la potencia en función de la carga que utilicen los dispositivos. En la tabla 5.4 se muestran los valores mínimos, máximos y las medias cuadráticas. Es reseñable

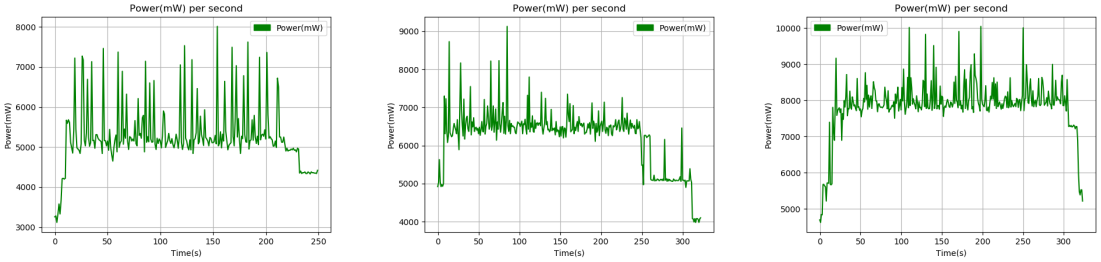
1 NCS2	Máximo: 2530 mW Mínimo: 1100 mW P_{RMS}: 1712.16 mW
2 NCS2	Máximo 4440 mW Mínimo: 1560 mW P_{RMS}: 2975.24 mW
3 NCS2	Máximo: 5080 mW Mínimo: 2410 mW P_{RMS}: 4215.40 mW

Tabla 5.4: Valores de potencia medidos de los NCS2 por separado para las 3 configuraciones del clúster.

1 NCS2	Máximo: 8010 mW Mínimo: 3120 mW P_{RMS}: 5302.10 mW
2 NCS2	Máximo: 9130 mW Mínimo: 3980 mW P_{RMS}: 6255.43 mW
3 NCS2	Máximo: 10050 mW Mínimo: 4630 mW P_{RMS}: 7883.98 mW

Tabla 5.5: Valores de potencia medidos de los NCS2 más la Raspberry Pi 3B+ para las 3 configuraciones del clúster.

que cada acelerador extra añadido al sistema supone un incremento de 1.2 watios. La tabla 5.5 permite resaltar que la configuración más "cara" de 3 NCS2 más la Raspberry, apenas asciende a 10W en sus picos máximos.



(a) 1 NCS2 (b) 2 NCS2 (c) 3 NCS2

Figura 5.5: Consumo de potencia del sistema propuesto con 1, 2 y 3 NCS2 más la Raspberry Pi 3B+.

5.2.2. Resultados en Raspberry Pi 4 como host de la aplicación

Una vez observados los resultados obtenidos en la Raspberry Pi 3B+ y, asumiendo el cuello de botella que supone la falta de USB 3.0 en esta placa, se procedió a tomar me-

didas sobre la nueva Raspberry Pi 4. Esta placa solventa definitivamente el problema de las comunicaciones y, además, añade mucha más potencia a nivel de procesador y memoria RAM.

En la figura 5.6 se muestran los resultados medidos en la Raspberry Pi 4 con cada una de las configuraciones del clúster de NCS2 propuesto. En ella, se puede observar que se consigue duplicar el rendimiento con respecto a su predecesora, la Raspberry Pi 3B+. No obstante, al igual que en esta última, no se puede apreciar un speedup significativo entre 2 y 3 NCS2.

Es importante resaltar que en estas condiciones, con la configuración de 2 NCS2, se consigue igualar el rendimiento del portátil aislado. Esto podría parecer un resultado exiguo, pero se debe tener en cuenta que el precio del sistema propuesto es aproximadamente un tercio del portátil y el consumo es cinco veces menor. Se han realizado las mismas pruebas de consumo de potencia sobre la Raspberry Pi 4, dando resultados nunca superiores a 11.2W.

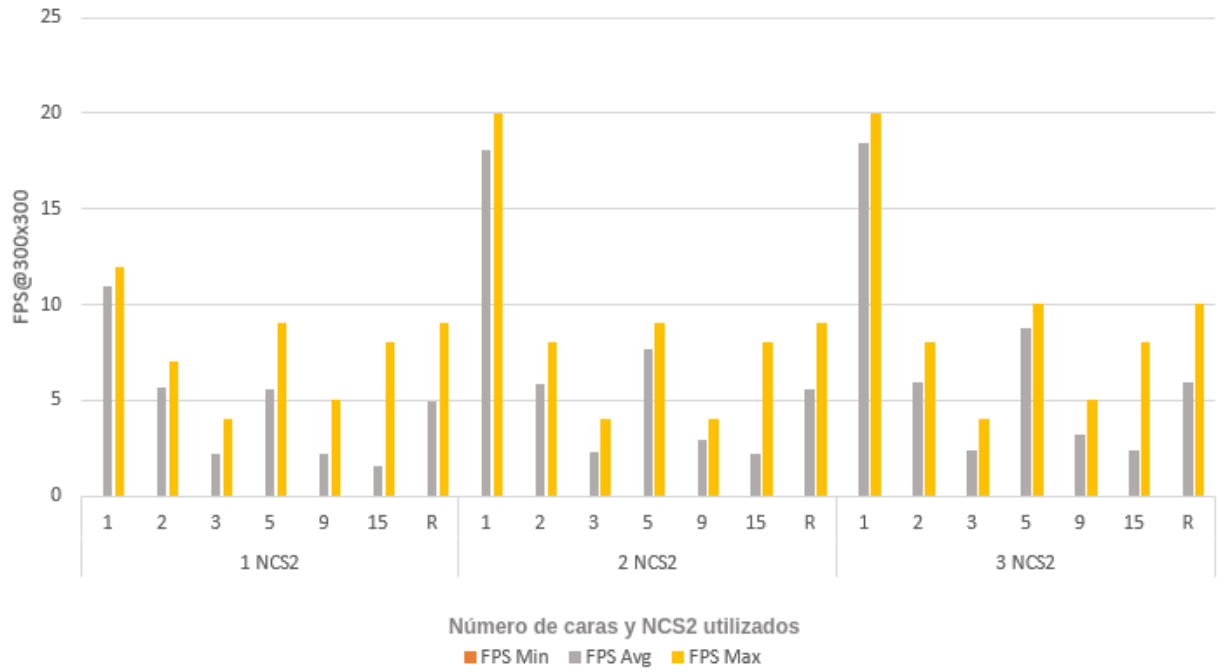


Figura 5.6: Resultados del experimento sobre una Raspberry 4 y 1, 2 y 3 NCS2.

En general, pese a no conseguir resultados de tiempo real, se consiguen 6 FPS@300x300 para el caso de caras aleatorias, lo cual implica un frame cada 166ms; suficiente para casos de uso como cámaras de seguridad en interiores.

5.3. Comparación de eficiencia

La comparación de los sistemas tradicionales con los sistemas empotrados suele llevar a dificultades y lagunas argumentales debido a que difieren enormemente no solo en rendimiento sino también en coste y consumo. Para realizar esta tarea, las medidas expresadas anteriormente de rendimiento (FPS@300x300) no sirven como métrica de comparación por lo que se ha utilizado una métrica de rendimiento/consumo para contrastar los resultados obtenidos en cada una de las plataformas previamente estudiadas.

Estos resultados se muestran en la tabla 5.6. En esta, se han omitido los resultados de los vídeos de 2 y 9 caras por motivos de simplificación y claridad en la presentación de los datos. Se debe resaltar que en el caso del PC, el consumo de potencia del i7 es de 37W, pero el máximo nivel de consumo de todo el dispositivo es de 57.9W según el benchmark 3DMark06[32]. Estos resultados han sido extraidos de este análisis detallado[33].

Como se puede observar en la tabla, el clúster formado por la Raspberry Pi 3B+ y el basado en el PC son cercanos en términos de FPS/Watio. Es reseñable que el primero es mejor en situaciones donde solo figura una sola cara mientras que el segundo se desempeña mucho mejor en escenarios multicara. En cualquier caso, los resultados de la Raspberry Pi 4 mejora a ambos en la mayoría de los casos.

Son reseñables los excelentes resultados alcanzados en los casos más extremos con 15 caras. La Raspberry Pi 4 con dos NCS2 equivalen al resultado obtenido por el portátil en el mismo supuesto con tres NCS2. Adicionalmente podemos observar que de forma general, la Raspberry Pi 4 duplica los resultados obtenidos por su predecesora erigiendose como la alternativa más eficiente en cuestión de rendimiento/potencia.

NCS2	Host	Potencia(Watio)	Caras	FPS@300x300	FPS/Watio
None	PC	57.9	1	21.84	0.37
			5	6.03	0.10
			15	1.78	0.030
			R	9.77	0.16
1	PC	59.1	1	17.83	0.30
			5	9.83	0.16
			15	4.71	0.07
			R	13.13	0.22
	Raspberry Pi 3B+	8.10	1	4.46	0.55
			5	2	0.24
			15	0.64	0.079
			R	2.11	0.26
	Raspberry Pi 4	8.8	1	11	1.25
			5	5.53	0.62
			15	1.51	0.18
			R	4.95	0.56
2	PC	60.3	1	30.42	0.50
			5	20.16	0.33
			15	9.38	0.15
			R	27.6	0.45
	Raspberry Pi 3B+	9.13	1	7.84	0.86
			5	3.97	0.43
			15	0.94	0.10
			R	2.64	0.28
	Raspberry Pi 4	10	1	18.06	1.80
			5	7.71	0.77
			15	2.16	0.21
			R	5.54	0.55
3	PC	61.5	1	42.4	0.68
			5	30.16	0.49
			15	13.5	0.21
			R	36.8	0.59
	Raspberry Pi 3B+	10.05	1	9.59	0.95
			5	3.97	0.39
			15	1.07	0.10
			R	2.92	0.29
	Raspberry Pi 4	11.2	1	18.4	1.64
			5	8.75	0.78
			15	2.39	0.41
			R	5.97	0.53

Tabla 5.6: Relación Rendimiento/Potencia del sistema propuesto con diferentes hosts y con respecto al número de NCS2 y caras por vídeo.

Capítulo 6

Conclusiones

Al inicio de este estudio se propusieron una serie de objetivos a resolver teniendo como fin principal mejorar el rendimiento de FaceNet en escenarios multicara para así demostrar la viabilidad de la escalabilidad horizontal de los aceleradores de la fase de inferencia sobre sistemas empotrados. El primero de ellos, consistía en el desarrollo de una aplicación de reconocimiento facial sobre Python y utilizando el toolkit Intel OpenVINO y los NCS2 como aceleradores de inferencia. En este apartado, los resultados han sido satisfactorios. Por un lado se ha conseguido realizar dicha aplicación y, por otro, se ha desarrollado una estructura modular para encapsular la lógica de las redes neuronales con el Inference Engine. De esta forma, no solo se da pie al reconocimiento facial, sino a la fácil implementación de otro tipo de redes neuronales sobre el toolkit.

El segundo objetivo consistía en desplegar dicha aplicación sobre diferentes plataformas hardware y tomar las medidas oportunas para analizar su desempeño. Este segundo objetivo también ha sido alcanzado con éxito ya que se han podido solventar los problemas que han ido surgiendo con las particularidades de cada una de las plataformas. Al final, se ha realizado el despliegue sobre un dispositivo basado en un Intel Core i7, una Rasperry Pi 3B+ y una Raspberry Pi 4. A continuación se han tomado las medidas de rendimiendo, consumo y escalabilidad.

- Las medidas de rendimiento se han medido utilizando vídeos de 300x300 píxeles y analizando los frames por segundo que la aplicación es capaz de inferir. Adicionalmente,

para evaluar el rendimiento, se han tenido en cuenta dos aproximaciones de paralelismo distintas, el multiproceso y el multihilo. Este hecho nos ha permitido observar directamente el efecto del GIL y, por tanto, las limitaciones de Python a nivel de programación paralela. Igualmente, ha permitido observar el rendimiento bruto de cada una de las plataformas evidenciando la superioridad de un sistema de alto consumo como es un portátil en comparación con las Raspberry.

- Las medidas de consumo se han tomado utilizando el INA260 como sensor de voltaje, corriente y potencia y una Raspberry Pi 3B+ independiente como dispositivo lector de las mediciones. Esto ha permitido medir el coste de un acelerador de las características del NCS2 mostrando el poco impacto energético que tienen este tipo de dispositivos. Además, las medidas de potencia permiten añadir una nueva dimensión a la comparativa que permite poner en contexto cada una de las plataformas en relación a su rendimiento.

- Las medidas de escalabilidad se han tomado en función del número de caras y número de aceleradores montados en arquitectura de clúster. Se ha evidenciado la disminución del rendimiento en función del número de caras presentes en cada frame y al mismo tiempo, la mejora de estos resultados en función del número de aceleradores. Especialmente interesante resulta el hecho de que el uso de más aceleradores aumenta la resiliencia del sistema ante los incrementos de carga mejorando así el rendimiento con mayor número de caras. Al mismo tiempo, se han comprobado problemas de escalabilidad en las plataformas empotradas donde no se ha apreciado un speedup significativo entre el uso de 2 y 3 aceleradores.

El tercer objetivo consistía en el análisis de los datos previamente obtenidos. Para comparar las plataformas de forma justa, hemos utilizado como métrica el FPS/Watio. El caso más claro que evidencia la necesidad de esta métrica es la comparación entre el portátil y la Raspberry Pi 3B+. El portátil como host alcanza rendimientos mucho mayores a los de

la Raspberry Pi 3B+, pero su consumo es mucho mayor. Por ello, hemos podido observar que diferencia que a priori nos marca el rendimiento bruto se estrecha sensiblemente cuando usamos el consumo como métrica adicional. Sin embargo, la superioridad del portátil decae cuando comparamos sus resultados con los obtenidos por la Raspberry Pi 4, la cual duplica el rendimiento de su predecesora y hace que su relación rendimiento/consumo sea mejor en la mayoría de los casos que las dos plataformas previamente analizadas.

Con este trabajo, se evidencia la utilidad clara de los aceleradores de la fase de inferencia como co-procesadores en dispositivos empotrados. Adicionalmente, se ha podido demostrar la viabilidad de escalar horizontalmente este tipo de dispositivos para paliar las caídas de rendimiento (escenarios multicara) y para permitir el despliegue de este tipo de aplicaciones sobre dispositivos de bajo costo. Es posible que actualmente no sean suficiente para aplicaciones de alta demanda pero son una opción seria que plantearse para el futuro del desarrollo de aplicaciones en la frontera. En este marco, su uso empotrado en cámaras de seguridad, coches, sistemas domóticos, etc. propicia la posibilidad de implementar soluciones de inteligencia artificial en escenarios IoT.

Capítulo 7

Trabajo Futuro

En este capítulo comentaremos las líneas abiertas que ha dejado este estudio y que podrían ser retomadas en futuros trabajos para seguir profundizando en el tema.

- Comparación entre tecnologías de la fase de inferencia. En este trabajo se ha hecho hincapié en el despliegue de soluciones utilizando los NCS2 de Intel y el toolkit OpenVINO. No obstante, sería muy interesante poder realizar una comparativa con sus competidores, la familia Nvidia Jetson y Google Coral. Especialmente interesante sería la comparación con Jetson dado que figuran como el paso intermedio entre una placa de uso general básico como la Raspberry Pi y un dispositivo de alto consumo como un Intel Core i7.

- Comparativa entre las APIs en Python y C++ del toolkit Intel OpenVINO. La aplicación desarrollada en este trabajo ha sido implementada utilizando la API en Python de OpenVINO. Por desgracia, los resultados obtenidos evidencian una clara limitación del lenguaje en cuanto al paralelismo de la aplicación. Esta problemática puede ser abordada directamente realizando nuevos experimentos directamente sobre la API de C++.

- Depliegue sobre enternos reales. Este estudio es eminentemente teórico y se ha realizado con pruebas en entornos controlados. Profundizando en este hecho, sería realmente interesante poder deplegar el sistema en escenarios reales y analizar el comportamiento

durante largos períodos de tiempo junto con el impacto enérgetico que pueda suponer en contraposición con otras alternativas presentes en el mercado.

Estas son algunas de las líneas de investigación que quedan abiertas en este estudio para futuros trabajos. No obstante, el escenario IoT y la inteligencia artificial avanza rápidamente por lo que es seguro que aparecerán nuevas alternativas a las presentadas en este trabajo. De esta forma, los resultados expuestos en este documento sirven como base para contextualizar las opciones contemporáneas y para cimentar la futura mejora y comparación de futuros nuevos sistemas.

Capítulo 8

Introduction

Historically, artificial intelligence (AI) applications were limited due to the computing power available at that moment. A proof of this statement is the existence of neural networks. The first definition of a perceptron showed up in 1958 by F.Rosenblatt[1]. It was defined as a unit that receives a variety of inputs and applies a mathematical function to them. Later, in 1965, the multilayer perceptron would be defined as well. However, even if the community was aware of the potential of this probabilistic model, it was discarded because of the huge quantity of data it has to be provided to the neural network in order to achieve a decent level of accuracy in its predictions.

Nowadays we do have the computational power necessary to manage those huge quantities of data. Investigation branches in Big Data (set of techniques that allows the efficient management of large quantities of data) and HPC (High Performance Computing) allowed the usage and exploitation of the data with AI algorithms.

In this point is where Machine Learning techniques and, more accurately, the Deep Learning ones get involved. Machine Learning is usually defined as the usage of algorithms to process data, extract features from them (process called training) with the goal of being able to predict those features over new data. In the other hand, Deep Learning is a subset of Machine Learning techniques. Typically, machine learning algorithms are combined in a variety of layers using each of them to highlight a certain feature of the provided data. Neural networks are a prominent model inside Deep Learning techniques.

This study will deepen in this branch of artificial intelligence and it will try to explore new solutions and perspectives that are currently in the market.

8.1. Motivation

The world is currently absorbed by the artificial intelligence. All companies and its products apply in one way or another some kind of artificial intelligence. Additionally, it is a concept that goes beyond the boundaries of the scientific community and gets to final consumers as a new revolutionary concept.

Technically speaking, the point in which a neural network is able to be trained with millions of data files is already achieved. It is also normal to achieve human accuracy prediction levels or even greater in some cases. This is the reason why the spotlight of the research efforts are not in the training process of neural networks (supported by the power of modern GPUs) but in the minimization of the training parameters and the acceleration of inference times instead.

Many of the new barriers of research in this scope, base its functionality in minimal response times. Until now, connecting to a cloud server was the most efficient solution to this problem. However, there were some situations in which event the small latency between the collector device and the cloud server was totally unassumable. An unexpected car crash is an example of this problematic.

At the same time, introducing inference in the embedded systems that collect data drift into an obvious performance limitation due to the low computational power that this kind of devices offer. Even with this consideration, it could still be enough for small and simple algorithms but it only get worse in scenarios like face recognition where the number of faces present in a single frame determines the overall performance of the application. What is more, in uncontrolled settings, the number of faces could escalate greatly. In this kind of applications, it is certain the need of adding some kind of co-processors in which delegate the calculation of the inference.

This is the reason why companies and research institutions are making great efforts to develop technologies that could accelerate the inference times of a neural network. This way, the algorithms could be directly deployed onto the data collector devices minimizing the latency to negligible times. In this work, one of this technologies will be chosen and it will be tested in order to check its limitations and performance in real case scenarios.

8.2. Milestones

This work aims to develop a face recognition application that would allow the measurements necessary in order to check the viability of horizontal scalability of low-cost devices. To do this, the performance, cost and consumption will be taken into account along its possibilities in edge computing use cases.

Below are shown the milestones set at the beginning of the study:

1. Develop a face recognition application using Python and Intel© OpenVINO™toolkit while accelerating the inference phase with various Intel© Neural Compute Stick 2.

2. Deploy the application over a general use laptop (based on Intel Core i7) and low-cost embedded systems like the Raspberry Pi 3B+ and Raspberry Pi 4. Take performance, consumption and scalability measures.

3. Analyze the data recorded previously in order to extract conclusions about the scope defined. Check the efficiency of the technologies chosen and its possibilities in edge computing scenarios facing more expensive technologies.

In order to do this, the following plan was designed exposed in the same order as the chapters of this document.

1. Investigate the state of the art in face recongition, neural networks and edge computing to guide the research and select the most suitable technique. This phase is explained in details in chapter 2.

2. Development of the hardware and software architecture. This pase is developed in chapter 3.

3. Once created the architecture, develop and application that could be deployed onto it using Python and the Intel OpenVINO toolkit. Detailed explanations can be read in chapter 4.

4. Take measurements and resume the results obtained along the conclusions that can be extracted. They are shown in chapter 5.

5. Finish the study detailing the conclusions extracted from the results and combining them with the state of the art. They are shown in chapter 6.

Along all the chapters of this document the problems faced during the development will be explained alongside their resolutions or palliation. To conclude this document, chapter 7 ponder about the future investigations in this field left opened after this study.

Capítulo 9

Conclusions

At the beginning of this work, a variety of milestones were proposed in order to achieve a better performance of FaceNet in multi-face scenarios. This way, it is possible to prove horizontal scalability of low-cost accelerators as a real alternative when deploying deep learning solutions over embedded systems. The first milestone consisted on developing a face recognition application using Python and the Intel OpenVINO toolkit, using the NCS2 to accelerate the inference phase. In this phase of the study, the results obtained have been satisfactory. In the one hand, the application has been developed successfully and, in the other hand, a modular structure has been developed in order to encapsulate the neural networks logic and interactions with the Inference Engine. By doing this, it does not only facilitates the face recognition task but it simplifies the implementation of any other neural network over the toolkit.

The second milestone consisted on deploying the application developed over different kind of platforms while taking measures of its performance. This task was completed successfully as well due to the resolution of the errors and problems that each platform (with its own particularities), presented while deploying the application. At last, the host devices used for the deployment were an Intel i7 based general purpose laptop, a Raspberry Pi 3B+ and a Raspberry Pi 4. Following this, the measurements of performance, consumption and scalability were taken.

- Performance measurements have been taken using 300x300 pixel videos and analyzing

the overall frame rate that the application is able to infer. Additionally, in order to evaluate the performance, two different parallel approaches have been used: the multithreading and the multiprocessing approach. This fact allowed to observe the GIL effect over the parallelization level and thus test the limitation of Python in terms of parallel programming. At the same time, the raw performance could be registered showing the superiority of a high-end system like an Intel i7 over the embedded boards like the Raspberry used in this study.

- Consumption metrics were taken using the INA260 power and current sensor and a Raspberry Pi 3B+ as an independent reading device. This allowed to check the cost of a single NCS2 accelerator showcasing the low energetic impact of this type of devices. They also add a new dimension to the comparative between the different host platforms, contextualizing the performance results obtained before and allowing a fair comparison among them.

- Scalability measurements have been recorded taking into account the number of faces and the number of NCS2 accelerators mounted in a cluster formation. Evidence of the performance downgrading have been found when increasing the number of faces in each frame and, at the same time, the increase of performance when scaling the number of accelerators. It must be highlighted that the more NCS2 used in the system, the better resilience it has when facing an increasement of workload. However, during the experiment, a problem in scalability have been found in the embedded platforms where there is not a significative speedup between the usage of 2 and 3 NCS2.

The final milestone consisted on the analysis of the previously recorded data. In order to assure a fair comparison between the platforms, a FPS/Watt metric have been used. This metric is necessary when comparing the Intel i7 based laptop with the embedded boards. The first one is clearly superior in terms of raw performance but its consumption is far greater. Adding the consumption dimension to the raw performance, it is visible that

differences between results narrow significatively although the laptop is still more efficient than the Raspberry Pi 3B+. Nevertheless, the superiority of the laptop drops down when comparing the results obtained with the ones of the Raspberry Pi 4, platform that doubles the performance of its predecessor and makes its performance/consumption ratio the best among the platforms used in the experiment.

With this study, the usefulness of inference accelerators as co-processor of embedded systems is proved. It also proves the horizontal scalability as a real alternative to improve the downgrade of performance in deep learning applications (ej. multiface scenario in face recogniton) and to allow the deployment of this kind of applications over low-cost devices. Possibly, this accelerators are not powerful enough to support high demanding scenarios but they have proven to be a serious alternative when talking about the future of edge computing. In this frame, its embedded usage in security cameras, smart cars, home automation systems, etc. favors the deployment of even more powerful artificial intelligence solutions in IoT scenarios.

Bibliografía

[1] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[2] Rina Dechter. Learning while searching in constraint-satisfaction-problems. pages 178–185, 01 1986.

[3] A. G. Ivakhnenko. Polynomial theory of complex systems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-1(4):364–378, Oct 1971.

[4] A.G. Ivakhnenko and V.G. Lapa. *Cybernetic Predicting Devices*. Jprs report. CCM Information Corporation, 1973.

[5] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[6] Cary Champlin, David Bell, and Celina Schocken. Ai medicine comes to africa's rural clinics. *IEEE Spectrum*, Apr. 2017.

[7] Raúl Murillo Montero, Alberto A. Del Barrio, and Guillermo Botella. Template-based posit multiplication for training and inferring in neural networks. *CoRR*, abs/1907.04091, 2019.

[8] Min Soo Kim, Alberto Del Barrio, Leonardo Oliveira, Roman Hermida, and Nader Bagherzadeh. Efficient mitchell's approximate log multipliers for convolutional neural networks. *IEEE Transactions on Computers*, PP:1–1, 11 2018.

[9] HyunJin Kim, Min Soo Kim, Alberto Del Barrio, and Nader Bagherzadeh. A cost-efficient iterative truncated logarithmic multiplication for convolutional neural networks. pages 108–111, 06 2019.

[10] Mei Wang and Weihong Deng. Deep face recognition: A survey. *ArXiv*, abs/1804.06655, 2018.

[11] Matthew Turk and Alex Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991. PMID: 23964806.

[12] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007.

[13] Lfw: Results. http://vis-www.cs.umass.edu/lfw/results.html. Accessed: 20-09-2019.

[14] Yaniv Taigman, Ming Wei Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, 2014.

[15] Yu Liu, Hongyang Li, and Xiaogang Wang. Learning deep features via congenerous cosine loss for person recognition, 2017.

[16] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[18] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. *CoRR*, abs/1503.03832, 2015.

[19] Tensorflow: An end-to-end open source machine learning platform. https://www.tensorflow.org/. Accessed: 19-09-2019.

[20] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, June 2015.

[21] Dave Evans. Internet de las cosas. Cómo la próxima evolución de Internet lo cambia todo. Technical report, Cisco Internet Business Solutions Group, 04 2011.

[22] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, Oct 2016.

[23] Intel© distribution of openvino™toolkit. https://software.intel.com/en-us/openvino-toolkit. Accessed: 19-09-2019.

[24] Intel© movidius™neural compute stick. https://movidius.github.io/ncsdk/ncs.html. Accessed: 29-08-2019.

[25] Intel© neural compute stick 2. https://software.intel.com/en-us/neural-compute-stick. Accessed: 29-08-2019.

[26] Google coral usb accelerator. https://coral.ai/docs/accelerator/datasheet/#power-specifications. Accessed: 11/12/2019.

[27] Mariano Hernández, Alberto Del Barrio, and Guillermo Botella. An ultra low-cost cluster based on low-end fpgas. page 20, 07 2018.

[28] P. Teodoro. Uso de tecnologías aplicadas de bajo coste en aplicaciones de deep learning. *UNED*, Feb. 2019.

[29] David Sandberg. facenet. https://github.com/davidsandberg/facenet, 2018.

[30] Shuo Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Wider face: A face detection benchmark. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[31] Thread state and the global interpreter lock. https://docs.python.org/3/c-api/init.html#thread-state-and-the-global-interpreter-lock. Accessed: 15-10-2019.

[32] https://benchmarks.ul.com/legacy-benchmarks. https://benchmarks.ul.com/legacy-benchmarks, 2019. Accessed: 14-11-2019.

[33] Review lenovo ideapad z510 notebook. https://www.notebookcheck.net/Review-Lenovo-IdeaPad-Z510-Notebook.105627.0.html, 2013. Accessed: 14-11-2019.